Business Case: Delhivery - Feature Engineering

About Delhivery

Delhivery is the largest and fastest-growing fully integrated player in India by revenue in
Fiscal 2021. They aim to build the operating system for commerce, through a combination of
world-class infrastructure, logistics operations of the highest quality, and cutting-edge
engineering and technology capabilities.
The Data team builds intelligence and capabilities using this data that helps them to widen the
gap between the quality, efficiency, and profitability of their business versus their competitors.

Problem Statement

The company wants to understand and process the data coming out of data engineering pipelines.
The data is not cleaned. So, there is need to clean, sanitize and manipulate data to get useful
features out of raw fields.
There is need to frame new features to conclude relevant information from dataset and hence, can
be helpful for data science team to build forecasting models on it.

In [236…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math, random
from scipy import stats

import warnings
warnings.filterwarnings("ignore")
```

In [236…

```
from collections import deque
from scipy.stats import norm
from scipy.stats import binom, poisson, expon, geom
from statsmodels.stats.weightstats import ztest
from scipy.stats import norm, ttest_ind, ttest_rel, ttest_1samp
from scipy.stats import chi2, chi2_contingency, chisquare
from scipy.stats import f_oneway as anova_1way

from scipy.stats import kruskal # numeric Vs catogerical
from scipy.stats import pearsonr, spearmanr # numeric Vs numeric
from scipy.stats import kstest # cdf
from scipy.stats import levene
from scipy.stats import norm
from scipy.stats import shapiro
```

```
from statsmodels.distributions.empirical_distribution import ECDF
from statsmodels.graphics.gofplots import qqplot, qqplot_2samples
from scipy.stats import skew, shapiro #[Test for normality]
```

In [236…
```
data = pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/
```

In [236…
```
data.head(2)
```

Out[2367]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_c
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38812
1	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38812

2 rows × 24 columns

Analyzing Structure of data

In [236…
```
# shape of dataset
data.shape

# It has 24 features or fields with 144867 entries.
```

Out[2368]:
```
(144867, 24)
```

In [236…
```
# Basic information about dataset
data.info()

# It can be observed the non-null count of some fields are less than 144867, this m
# missing values in that field.
# The datatypes included majorly in dataset are object and float64.
# Only field "is_cutoff" is of datatype bool,
# only field "cutoff_factor" is of datatype integer.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 24 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   data                            144867 non-null  object 
 1   trip_creation_time              144867 non-null  object 
 2   route_schedule_uuid             144867 non-null  object 
 3   route_type                      144867 non-null  object 
 4   trip_uuid                       144867 non-null  object 
 5   source_center                   144867 non-null  object 
 6   source_name                     144574 non-null  object 
 7   destination_center              144867 non-null  object 
 8   destination_name                144606 non-null  object 
 9   od_start_time                   144867 non-null  object 
 10  od_end_time                     144867 non-null  object 
 11  start_scan_to_end_scan          144867 non-null  float64
 12  is_cutoff                       144867 non-null  bool   
 13  cutoff_factor                   144867 non-null  int64  
 14  cutoff_timestamp                144867 non-null  object 
 15  actual_distance_to_destination  144867 non-null  float64
 16  actual_time                     144867 non-null  float64
 17  osrm_time                       144867 non-null  float64
 18  osrm_distance                   144867 non-null  float64
 19  factor                          144867 non-null  float64
 20  segment_actual_time             144867 non-null  float64
 21  segment_osrm_time               144867 non-null  float64
 22  segment_osrm_distance           144867 non-null  float64
 23  segment_factor                  144867 non-null  float64
dtypes: bool(1), float64(10), int64(1), object(12)
memory usage: 25.6+ MB
```

In [237…

```
# unique values of all fields of dataset
data.nunique(axis=0)

# fields such as "data","route_type" and "is_cutoff" have only two types of values
```

Out[2370]:

```
data                                2
trip_creation_time              14817
route_schedule_uuid              1504
route_type                          2
trip_uuid                       14817
source_center                    1508
source_name                      1498
destination_center               1481
destination_name                 1468
od_start_time                   26369
od_end_time                     26369
start_scan_to_end_scan           1915
is_cutoff                           2
cutoff_factor                     501
cutoff_timestamp                93180
actual_distance_to_destination  144515
actual_time                      3182
osrm_time                        1531
osrm_distance                  138046
factor                          45641
segment_actual_time               747
segment_osrm_time                 214
segment_osrm_distance          113799
segment_factor                   5675
dtype: int64
```

In [237…
```
# Checking for duplicated values
data.duplicated().sum()

# There are no duplicate entries in the dataset
```

Out[2371]:
```
0
```

Analyzing null values and handling them

In [237…
```
# calculating null values in fields
data_missing_value = pd.DataFrame(data.isnull().sum()).reset_index()
data_missing_value.rename({"index":"fields",0:"Count_of_missing_values"},axis = 1,
data_missing_value

# There are 293 missing values in "source_name" field
# There are 261 missing values in "destination" field
```

Out[2372]:

	fields	Count_of_missing_values
0	data	0
1	trip_creation_time	0
2	route_schedule_uuid	0
3	route_type	0
4	trip_uuid	0
5	source_center	0
6	source_name	293
7	destination_center	0
8	destination_name	261
9	od_start_time	0
10	od_end_time	0
11	start_scan_to_end_scan	0
12	is_cutoff	0
13	cutoff_factor	0
14	cutoff_timestamp	0
15	actual_distance_to_destination	0
16	actual_time	0
17	osrm_time	0
18	osrm_distance	0
19	factor	0
20	segment_actual_time	0
21	segment_osrm_time	0
22	segment_osrm_distance	0
23	segment_factor	0

Analysis of fields Source_name and destination_name having missing values

In [237…
```
# Analysis of "source_name" field in relation with "source_center" field
index_var = data[data["source_name"].isna()].index
data["source_center"].iloc[index_var].unique()

# Below are the source_centers whose source_names are not known.
```

Out[2373]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND509103AAC', 'IND126116AAA', 'IND331022A1B',
       'IND505326AAB', 'IND852118A1B'], dtype=object)
```

In [237…
```
# Analysis of "destination_name" field in relation with "destination_center" field
index_var = data[data["destination_name"].isna()].index
data["destination_center"].iloc[index_var].unique()

# Below are the source_centers whose destination_names are not known.
```

Out[2374]:
```
array(['IND342902A1B', 'IND577116AAA', 'IND282002AAD', 'IND465333A1B',
       'IND841301AAC', 'IND505326AAB', 'IND852118A1B', 'IND126116AAA',
       'IND509103AAC', 'IND221005A1A', 'IND250002AAC', 'IND331001A1C',
       'IND122015AAC'], dtype=object)
```

Above are two arrays

1. Above array 1 shows the center id of source whose names are missing. 10 unique source_names are missing
2. Above array 2 shows the center id of destination whose names are missing. 13 unique destination_names are missing.
3. If closely observed, 10 source centers are already included in 13 destination centers.
4. Centers id can be source ordestination.

Based on above points, solution will be:

1. Assign names to all distinct center id
2. Fill all nan values of names with names with refernce to center ids of source and destination.

Filling missing values

In [237…
```
# Handling missing values
name_dict = {'IND342902A1B':'center_name_1 (S1)', 'IND577116AAA':'center_name_2 (S
             'IND282002AAD':'center_name_3 (S3)', 'IND465333A1B':'center_name_4 (S
             'IND841301AAC':'center_name_5 (S5)', 'IND505326AAB':'center_name_6 (S
             'IND852118A1B':'center_name_7 (S7)', 'IND126116AAA':'center_name_8 (S
             'IND509103AAC':'center_name_9 (S9)', 'IND221005A1A':'center_name_10 (
             'IND250002AAC':'center_name_11 (S11)', 'IND331001A1C':'center_name_12
             'IND122015AAC':'center_name_13 (S13)', 'IND331022A1B':'center_name_14
data["source_name"] = data["source_name"].fillna(data["source_center"].map(name_di
data["destination_name"] = data["destination_name"].fillna(data["destination_cente

#Check again missing values
data.isnull().sum().sum()

# No missing values in dataset
```

Out[2375]:
```
0
```

Analysis and treatment of Unknown fields

Fields like
"is_cutoff","cutoff_factor","cutoff_timestamp","factor" and
"segment_factor"
are not known.
But if closely observed:
1. "factor" field has been derived from fields "actual_time" and
"osrm_time".
    It has been obtained as "actual_time"/"osrm_time".
2. "segment_factor" has been derived from fields
"segment_actual_time" and "segment_osrm_time"
    It has been obtained as
"segment_actual_time"/"segment_osrm_time".
3. "is_cutoff","cutoff_factor" and "cutoff_timestamp" fields are
still unknown. So, it is better to
    drop those fields

In [237…
```
# dropping columns "is_cutoff","cutoff_factor" and "cutoff_timestamp" from dataset
data.drop(columns=["is_cutoff","cutoff_factor","cutoff_timestamp"], inplace=True)
```

In [237…
```
#checking dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 21 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            144867 non-null  object
 1   trip_creation_time              144867 non-null  object
 2   route_schedule_uuid             144867 non-null  object
 3   route_type                      144867 non-null  object
 4   trip_uuid                       144867 non-null  object
 5   source_center                   144867 non-null  object
 6   source_name                     144867 non-null  object
 7   destination_center              144867 non-null  object
 8   destination_name                144867 non-null  object
 9   od_start_time                   144867 non-null  object
 10  od_end_time                     144867 non-null  object
 11  start_scan_to_end_scan          144867 non-null  float64
 12  actual_distance_to_destination  144867 non-null  float64
 13  actual_time                     144867 non-null  float64
 14  osrm_time                       144867 non-null  float64
 15  osrm_distance                   144867 non-null  float64
 16  factor                          144867 non-null  float64
 17  segment_actual_time             144867 non-null  float64
 18  segment_osrm_time               144867 non-null  float64
 19  segment_osrm_distance           144867 non-null  float64
 20  segment_factor                  144867 non-null  float64
dtypes: float64(10), object(11)
memory usage: 23.2+ MB
```

Converting datatypes of features in appropriate datatype

In [237…
```
#converting all date and time fields datatypes with appropriate datatype
data[["od_start_time", "od_end_time",'trip_creation_time']] = data[["od_start_time"
                                                                   'trip_creation_time']].app
```

```
# It can be observed that fields like "actual_time","osrm_time","segment_actual_ti
# "segment_osrm_time" denotes time but as it is not clear from dataset, whether th
# in hours or minutes.
# It is therefore, concluded not to convert datatype of these fields.
```

In [237…

```
# Analyzing source_name and destination_name features
data[["source_name","destination_name"]].head(3)

# It can be observed that City name has been written along with district name.
# To extract city name, the datatype of these fields must be string.
```

Out[2379]:

	source_name	destination_name
0	Anand_VUNagar_DC (Gujarat)	Khambhat_MotvdDPP_D (Gujarat)
1	Anand_VUNagar_DC (Gujarat)	Khambhat_MotvdDPP_D (Gujarat)
2	Anand_VUNagar_DC (Gujarat)	Khambhat_MotvdDPP_D (Gujarat)

In [238…

```
data["source_name"]= data["source_name"].astype("str")
data["destination_name"] = data["destination_name"].astype("str")

#check again datatypes of dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 144867 entries, 0 to 144866
Data columns (total 21 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   data                            144867 non-null  object
 1   trip_creation_time              144867 non-null  datetime64[ns]
 2   route_schedule_uuid             144867 non-null  object
 3   route_type                      144867 non-null  object
 4   trip_uuid                       144867 non-null  object
 5   source_center                   144867 non-null  object
 6   source_name                     144867 non-null  object
 7   destination_center              144867 non-null  object
 8   destination_name                144867 non-null  object
 9   od_start_time                   144867 non-null  datetime64[ns]
 10  od_end_time                     144867 non-null  datetime64[ns]
 11  start_scan_to_end_scan          144867 non-null  float64
 12  actual_distance_to_destination  144867 non-null  float64
 13  actual_time                     144867 non-null  float64
 14  osrm_time                       144867 non-null  float64
 15  osrm_distance                   144867 non-null  float64
 16  factor                          144867 non-null  float64
 17  segment_actual_time             144867 non-null  float64
 18  segment_osrm_time               144867 non-null  float64
 19  segment_osrm_distance           144867 non-null  float64
 20  segment_factor                  144867 non-null  float64
dtypes: datetime64[ns](3), float64(10), object(8)
memory usage: 23.2+ MB
```

Feature Engineering

Study of delivery details of one package: trip_uuid = "trip-153671041653548748"

In [238…

```
# Analyzing all fields and values for trip-153671041653548748
data_temp = data[data["trip_uuid"] == "trip-153671041653548748"].iloc[18:23,:]
data_temp.loc[:,["source_name","destination_name","od_start_time","od_end_time",
                 "start_scan_to_end_scan","actual_time","osrm_time","osrm_distance"
                 "factor","segment_actual_time","segment_osrm_time",
                 "segment_osrm_distance","segment_factor"]]

# displaying only the relevant cells/blocks of the table for trip-1536710416535487
```

Out[2381]:

	source_name	destination_name	od_start_time	od_end_time	start_scan_to_en
124999	Bhopal_Trnsport_H (Madhya Pradesh)	Kanpur_Central_H_6 (Uttar Pradesh)	2018-09-12 00:00:16.535741	2018-09-12 16:39:46.858469	
125000	Bhopal_Trnsport_H (Madhya Pradesh)	Kanpur_Central_H_6 (Uttar Pradesh)	2018-09-12 00:00:16.535741	2018-09-12 16:39:46.858469	
125001	Bhopal_Trnsport_H (Madhya Pradesh)	Kanpur_Central_H_6 (Uttar Pradesh)	2018-09-12 00:00:16.535741	2018-09-12 16:39:46.858469	
125002	Kanpur_Central_H_6 (Uttar Pradesh)	Gurgaon_Bilaspur_HB (Haryana)	2018-09-12 16:39:46.858469	2018-09-13 13:40:23.123744	
125003	Kanpur_Central_H_6 (Uttar Pradesh)	Gurgaon_Bilaspur_HB (Haryana)	2018-09-12 16:39:46.858469	2018-09-13 13:40:23.123744	

It can be observed that this package was packed at Bhopal(madhya pradesh) and delivered to Gurgaon_Bilaspur (Haryana) via Kanpur(Uttar Pradesh)

1. order start time and end time is different for one center(source) to other center(destination)
2. difference of start time and end time (given in field start scan to end scan) is also different.
3. Actual time and osrm (open source routing machine) time have been increasing with rows for Bhopal to Kanpur. Similar trend can be observed for Kanpur to Gurgaon.
4. osrm (open source routing machine) distance has been increasing with rows for Bhopal to Kanpur and then from Kanpur to Gurgaon.
5. segment_actual_time, segment_osrm_time, segment_osrm_distance are also changing with rows.

conclusion: what steps must be done

1. The dataset data_temp (for trip-153671041653548748) needs to be grouped by source_name and destination name
2. Cumulative sum needs to be done for fields like segment_actual_time, segment_osrm_time, segment_osrm_distance
3. For Actual time, osrm time and osrm distance, there is need to extract first and last value grouped by source and destination names.
4. factor needs to be aggregated as first and last
5. segment factor needs to be aggregated as cumulative sum.

Merging of rows and aggregation of fields

A1. Pre-processing : merging the rows group by trip_uuid, source_center, destination_center

In [238…

```
# using original dataset
data.head(1)
```

Out[2382]:

	data	trip_creation_time	route_schedule_uuid	route_type	trip_uuid	source_c
0	training	2018-09-20 02:35:36.476840	thanos::sroute:eb7bfc78-b351-4c0e-a951-fa3d5c3...	Carting	trip-153741093647649320	IND38812

1 rows × 21 columns

1. Cumulative sum of fields: segment_actual_time, segment_osrm_time, segment_osrm_distance

In [238…

```
# calculating cumulative sum in relevant fields
# group by trip_uuid, source center and destination center
data['data_feature'] = data['trip_uuid'] + data['source_center'] + data['destinati
segment_fields = ['segment_actual_time','segment_osrm_distance','segment_osrm_time
for columns in segment_fields:
    data[columns + '_sum'] = data.groupby('data_feature')[columns].cumsum()
```

2. Extracting first and last values from relevant fields

In [238…

```
# calculating first and last values in relevant fields
# group by trip_uuid, source center and destination center
features_dict = { 'data'                    :'first',
                  'trip_creation_time'      :'first',
                  'route_schedule_uuid'     :'first',
                  'route_type'              :'first',
                  'trip_uuid'               :'first',
                  'source_center'           :'first',
                  'source_name'             :'first',
                  'destination_center'      :'last',
                  'destination_name'        :'last',
                  'od_start_time'           :'first',
                  'od_end_time'             :'first',
                  'start_scan_to_end_scan':'first',
          'actual_distance_to_destination':'last',
                'actual_time'               :'last',
                'osrm_time'                 :'last',
                'osrm_distance'             :'last',
                 'factor'                   :'last',
                 'segment_factor'           :'last',
                'segment_actual_time_sum'  :'last',
               'segment_osrm_distance_sum':'last',
                'segment_osrm_time_sum'    :'last'  }


data_feature = data.groupby('data_feature').agg(features_dict).reset_index()
data_feature = data_feature.sort_values(by=['data_feature','od_end_time'], ascendi

# we obtained our new dataset
```

New Dataset- grouped by trip_uuid,source and destination center

In [238…
```
# Analysis of new dataset
data_feature.shape
```

Out[2385]: (26368, 22)

In [238…
```
# displaying all features/fields of new dataset
display(data_feature.iloc[:,:4].head(4))
display(data_feature.iloc[:,4:10].head(4))
display(data_feature.iloc[:,10:17].head(4))
display(data_feature.iloc[:,17:].head(4))
```

	data_feature	data	trip_creation_time	route_schedule_
0	trip-153671041653548748IND209304AAAIND000000ACB	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c98 a29b-4a0b-b 288c
1	trip-153671041653548748IND462022AAAIND209304AAA	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c98 a29b-4a0b-b 288c
2	trip-153671042288605164IND561203AABIND562101AAA	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0 bb0b-4c53-8 eb2a
3	trip-153671042288605164IND572101AAAIND561203AAB	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0 bb0b-4c53-8 eb2a

	route_type	trip_uuid	source_center	source_name	destination_center	
0	FTL	trip-153671041653548748	IND209304AAA	Kanpur_Central_H_6 (Uttar Pradesh)	IND000000ACB	
1	FTL	trip-153671041653548748	IND462022AAA	Bhopal_Trnsport_H (Madhya Pradesh)	IND209304AAA	
2	Carting	trip-153671042288605164	IND561203AAB	Doddablpur_ChikaDPP_D (Karnataka)	IND562101AAA	
3	Carting	trip-153671042288605164	IND572101AAA	Tumkur_Veersagr_I (Karnataka)	IND561203AAB	Dc

	od_start_time	od_end_time	start_scan_to_end_scan	actual_distance_to_destination	actual_tir
0	2018-09-12 16:39:46.858469	2018-09-13 13:40:23.123744	1260.0	383.759164	73
1	2018-09-12 00:00:16.535741	2018-09-12 16:39:46.858469	999.0	440.973689	83
2	2018-09-12 02:03:09.655591	2018-09-12 03:01:59.598855	58.0	24.644021	4
3	2018-09-12 00:00:22.886430	2018-09-12 02:03:09.655591	122.0	48.542890	9

	factor	segment_factor	segment_actual_time_sum	segment_osrm_distance_sum	segment_osrm
0	2.224924	2.000000	728.0	670.6205	
1	2.139175	7.333333	820.0	649.8528	
2	1.807692	2.142857	46.0	28.1995	
3	2.285714	6.666667	95.0	55.9899	

3. Extract features from the fields

a. Functions to split source/destination name into City-place-code(State)

In [238...

```
# Function to split State from name
def name_state(x):
    state = x.split('(')[-1]    #splitting from '(' and take index 1 value
    state = state.split(')')[0]
    return state         #removing ')' from ending

# Function to split code from name
def name_code(x):
    code = x.split(' (')[0]    #splitting from '(' and take index 0 value
    if len(code.split('_')) >= 2 and (code.split('_')[-1].isupper() or code.split(
        return code.split('_')[-1]
    code1 = code.split(' ')
    if len(code1) >= 2 and code1[-1].isupper():
        return code1[-1]
    else:
        return 'none'

# Function to split city from name
def name_city(x):
    city = x.split(' (')[0]    #splitting from '(' and take index 0 value
    return city.split('_')[0]

# function to split place from name
def name_place(x):
    place = x.split(' (')[0]    #splitting from '(' and take index 0 value
    place = place.split('_')
    if len(place) >= 2:
        return place[1]
    else:
        return place[0]
```

b. Splitting source name into City_place_code (State)

In [238...

```
# Splitting source name into City_place_code (State)
data_feature['source_state'] = data_feature['source_name'].apply(lambda x: name_st
data_feature['source_city']  = data_feature['source_name'].apply(lambda x: name_ci
data_feature['source_place'] = data_feature['source_name'].apply(lambda x: name_pl
data_feature['source_code']  = data_feature['source_name'].apply(lambda x: name_co
```

c. Splitting destination name into City_place_code (State)

In [238...

```
# Splitting destination name into City_place_code (State)
data_feature['destination_state'] = data_feature['destination_name'].apply(lambda
```

```
data_feature['destination_city']  = data_feature['destination_name'].apply(lambda x
data_feature['destination_place'] = data_feature['destination_name'].apply(lambda x
data_feature['destination_code']  = data_feature['destination_name'].apply(lambda x
```

d. Trip_creation_time: Extract features like month, year and day etc

In [239…
```
#extracting day, month & year from trip_creation_time
data_feature['trip_creation_year']=data_feature['trip_creation_time'].dt.year
data_feature['trip_creation_month']=data_feature['trip_creation_time'].dt.month
data_feature['trip_creation_day']=data_feature['trip_creation_time'].dt.day
```

In [239…
```
# checking missing terms again after extracting features
data_feature.isna().sum().sum()
```

Out[2391]:
```
0
```

In [239…
```
# displaying all features/fields of new dataset
display(data_feature.iloc[:,:4].head(4))
display(data_feature.iloc[:,4:10].head(4))
display(data_feature.iloc[:,10:17].head(4))
display(data_feature.iloc[:,17:25].head(4))
display(data_feature.iloc[:,25:].head(4))
```

	data_feature	data	trip_creation_time	route_schedule_
0	trip-153671041653548748IND209304AAAIND000000ACB	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c98 a29b-4a0b-b 288c
1	trip-153671041653548748IND462022AAAIND209304AAA	training	2018-09-12 00:00:16.535741	thanos::sroute:d7c98 a29b-4a0b-b 288c
2	trip-153671042288605164IND561203AABIND562101AAA	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0 bb0b-4c53-8 eb2a
3	trip-153671042288605164IND572101AAAIND561203AAB	training	2018-09-12 00:00:22.886430	thanos::sroute:3a1b0 bb0b-4c53-8 eb2a

	route_type	trip_uuid	source_center	source_name	destination_center	
0	FTL	trip-153671041653548748	IND209304AAA	Kanpur_Central_H_6 (Uttar Pradesh)	IND000000ACB	
1	FTL	trip-153671041653548748	IND462022AAA	Bhopal_Trnsport_H (Madhya Pradesh)	IND209304AAA	
2	Carting	trip-153671042288605164	IND561203AAB	Doddablpur_ChikaDPP_D (Karnataka)	IND562101AAA	
3	Carting	trip-153671042288605164	IND572101AAA	Tumkur_Veersagr_I (Karnataka)	IND561203AAB	Do

	od_start_time	od_end_time	start_scan_to_end_scan	actual_distance_to_destination	actual_tir
0	2018-09-12 16:39:46.858469	2018-09-13 13:40:23.123744	1260.0	383.759164	732
1	2018-09-12 00:00:16.535741	2018-09-12 16:39:46.858469	999.0	440.973689	830
2	2018-09-12 02:03:09.655591	2018-09-12 03:01:59.598855	58.0	24.644021	47
3	2018-09-12 00:00:22.886430	2018-09-12 02:03:09.655591	122.0	48.542890	96

	factor	segment_factor	segment_actual_time_sum	segment_osrm_distance_sum	segment_osrm
0	2.224924	2.000000	728.0	670.6205	
1	2.139175	7.333333	820.0	649.8528	
2	1.807692	2.142857	46.0	28.1995	
3	2.285714	6.666667	95.0	55.9899	

	source_code	destination_state	destination_city	destination_place	destination_code	trip_creatio
0	6	Haryana	Gurgaon	Bilaspur	HB	
1	H	Uttar Pradesh	Kanpur	Central	6	
2	D	Karnataka	Chikblapur	ShntiSgr	D	
3	I	Karnataka	Doddablpur	ChikaDPP	D	

The above data has been framed on basis of trip id, source id, and destination id.

Clean the data by deleting irrelevant columns for better analysis. Simultaneously, make a copy of original dataset

In [239...
```
data_copy = data_feature.copy(deep=True)
```

In [239...
```
data_feature.drop(columns=['data_feature', 'route_schedule_uuid','segment_factor',
```

In [239...
```
# displaying all features/fields of new dataset
display(data_feature.iloc[:,:8].head(4))
display(data_feature.iloc[:,8:16].head(4))
display(data_feature.iloc[:,16:24].head(4))
display(data_feature.iloc[:,24:].head(4))
```

	data	trip_creation_time	route_type	trip_uuid	source_center	source_n
0	training	2018-09-12 00:00:16.535741	FTL	trip-153671041653548748	IND209304AAA	Kanpur_Central (Uttar Prac
1	training	2018-09-12 00:00:16.535741	FTL	trip-153671041653548748	IND462022AAA	Bhopal_Trnspo (Madhya Prac
2	training	2018-09-12 00:00:22.886430	Carting	trip-153671042288605164	IND561203AAB	Doddablpur_ChikaDI (Karna
3	training	2018-09-12 00:00:22.886430	Carting	trip-153671042288605164	IND572101AAA	Tumkur_Veers (Karna

	od_start_time	od_end_time	start_scan_to_end_scan	actual_distance_to_destination	actual_tir
0	2018-09-12 16:39:46.858469	2018-09-13 13:40:23.123744	1260.0	383.759164	732
1	2018-09-12 00:00:16.535741	2018-09-12 16:39:46.858469	999.0	440.973689	830
2	2018-09-12 02:03:09.655591	2018-09-12 03:01:59.598855	58.0	24.644021	47
3	2018-09-12 00:00:22.886430	2018-09-12 02:03:09.655591	122.0	48.542890	96

	segment_osrm_distance_sum	segment_osrm_time_sum	source_state	source_city	source_place	s
0	670.6205	534.0	Uttar Pradesh	Kanpur	Central	
1	649.8528	474.0	Madhya Pradesh	Bhopal	Trnsport	
2	28.1995	26.0	Karnataka	Doddablpur	ChikaDPP	
3	55.9899	39.0	Karnataka	Tumkur	Veersagr	

	destination_place	destination_code	trip_creation_year	trip_creation_month	trip_creation_day
0	Bilaspur	HB	2018	9	12
1	Central	6	2018	9	12
2	ShntiSgr	D	2018	9	12
3	ChikaDPP	D	2018	9	12

A2. Pre-processing :further merging rows based on trip uuid

1. Cumulative sum of fields: 'actual_time', 'osrm_time', 'osrm_distance', segment_actual_time, segment_osrm_time, segment_osrm_distance

In [239...

```
# calculating cumulative sum in above fields
# group by trip_uuid
data_feature['df_feat'] = data_feature['trip_uuid']
```

```
segment_fields = ['start_scan_to_end_scan','actual_distance_to_destination','actua
                  'osrm_distance','segment_actual_time_sum','segment_osrm_time_sum',
for columns in segment_fields:
    data_feature[columns + '_sum'] = data_feature.groupby('df_feat')[columns].cums
```

2. Extracting first and last values from relevant fields

In [239…

```
# calculating first and last values in relevant fields
# group by trip_uuid, source center and destination center
df_dict = {'data':'first',  'trip_creation_time' :'first',
           'route_type':'first', 'trip_uuid'  :'first',
           'source_center':'last', 'source_name' :'last','source_state':'last',
          'source_city':'last', 'source_place':'last', 'source_code':'last',
           'destination_center' :'first', 'destination_name':'first', 'destinatio
          'destination_city':'first', 'destination_place':'first','destination_co
           'od_start_time' :'last', 'od_end_time':'first',
           'start_scan_to_end_scan_sum':'last', 'actual_distance_to_destination_s
           'actual_time_sum' :'last','osrm_time_sum' :'last', 'osrm_distance_sum'
           'segment_actual_time_sum_sum' :'last','segment_osrm_distance_sum_sum':
           'segment_osrm_time_sum_sum'   :'last',
          'trip_creation_year':'first','trip_creation_month':'first',
          'trip_creation_day':'first' }


df_feature = data_feature.groupby('df_feat').agg(df_dict).reset_index()
df_feature = df_feature.sort_values(by=['trip_uuid','od_end_time'], ascending=True
```

New Dataset- grouped by trip_uuid with all aggregations on fields

In [239…

```
df_feature.shape
```

Out[2398]: (14817, 30)

In [239…

```
# displaying all features/fields of new dataset
display(df_feature.iloc[:,:8].head(2))
display(df_feature.iloc[:,8:17].head(2))
display(df_feature.iloc[:,17:24].head(2))
display(df_feature.iloc[:,24:].head(2))
```

	df_feat	data	trip_creation_time	route_type	trip_uuid	source_cente
0	trip-153671041653548748	training	2018-09-12 00:00:16.535741	FTL	trip-153671041653548748	IND462022AA
1	trip-153671042288605164	training	2018-09-12 00:00:22.886430	Carting	trip-153671042288605164	IND572101AA

	source_city	source_place	source_code	destination_center	destination_name	destination_sta
0	Bhopal	Trnsport	H	IND000000ACB	Gurgaon_Bilaspur_HB (Haryana)	Harya
1	Tumkur	Veersagr	I	IND562101AAA	Chikblapur_ShntiSgr_D (Karnataka)	Karnata

	od_start_time	od_end_time	start_scan_to_end_scan_sum	actual_distance_to_destination_sum
0	2018-09-12 00:00:16.535741	2018-09-13 13:40:23.123744	2259.0	824.732854
1	2018-09-12 00:00:22.886430	2018-09-12 03:01:59.598855	180.0	73.186911

	segment_actual_time_sum_sum	segment_osrm_distance_sum_sum	segment_osrm_time_sum_sum
0	1548.0	1320.4733	1008.0
1	141.0	84.1894	65.0

Univariate and Bivariate Analysis

In [240…

```
#Analysis data of "data" column
d1 = pd.DataFrame(data_feature["data"].value_counts().reset_index())
d1.columns = ["data","Count"]
d1["% count"] = round((d1["Count"]/len(data_feature))*100,2)

#Analysis data of "route_type" column
d2 = pd.DataFrame(data_feature["route_type"].value_counts().reset_index())
d2.columns = ["route_type","Count"]
d2["% count"] = round((d2["Count"]/len(data_feature))*100,2)

#Analysis data of "month" column
d3 = pd.DataFrame(data_feature["trip_creation_month"].value_counts().reset_index()
d3.columns = ["month","Count"]
d3["% count"] = round((d3["Count"]/len(data_feature))*100,2)

display(d1)
display(d2)
display(d3)
```

	data	Count	% count
0	training	18947	71.86
1	test	7421	28.14

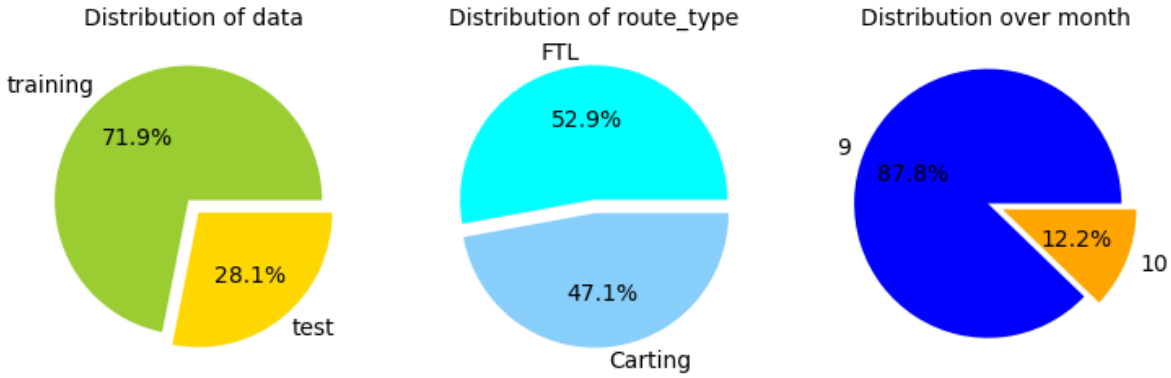
	route_type	Count	% count
0	FTL	13939	52.86
1	Carting	12429	47.14

	month	Count	% count
0	9	23159	87.83
1	10	3209	12.17

In [240…

```
# plotting distribution of categorical variables
plt.figure(figsize = (9,6))
explode_data = (0.06,0.06)
explode_route_type = (0.05,0.05)
explode_month = (0.06,0.06)
colors_data = ['yellowgreen','gold']
colors_route_type = ["cyan", 'lightskyblue']
colors_month = ["blue", 'orange']
```

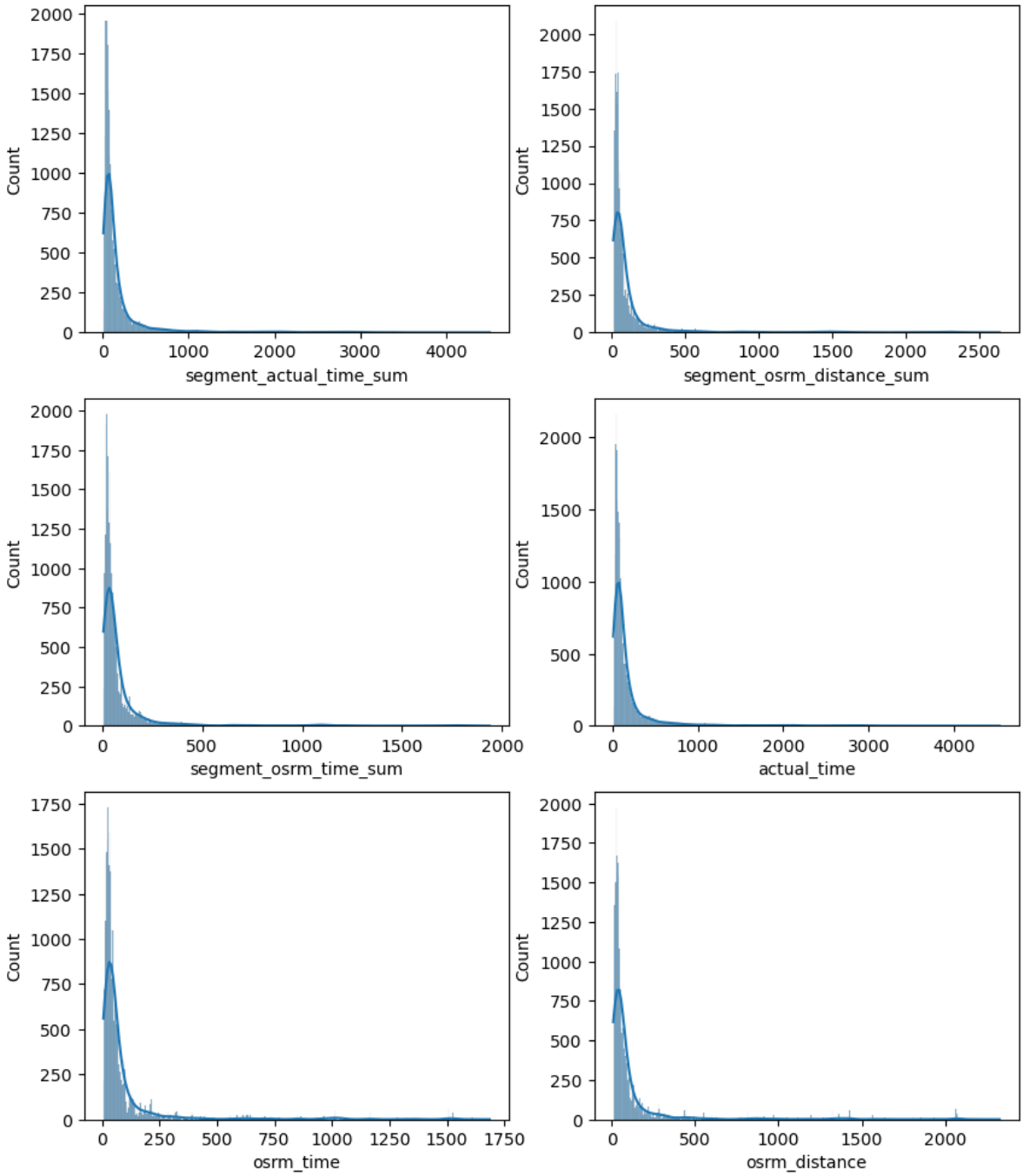
```
plt.subplot(1,3,1)
plt.pie(d1["Count"], labels=d1["data"], explode=explode_data,colors = colors_data,
        autopct='%1.1f%%')
plt.title("Distribution of data", fontsize = 10)

plt.subplot(1,3,2)
plt.pie(d2["Count"], labels=d2["route_type"], explode=explode_route_type,
        colors = colors_route_type, autopct='%1.1f%%')
plt.title("Distribution of route_type", fontsize = 10)

plt.subplot(1,3,3)
plt.pie(d3["Count"], labels=d3["month"], explode=explode_month,
        colors = colors_month, autopct='%1.1f%%')
plt.title("Distribution over month", fontsize = 10)
plt.show()
```



Univariate analysis of numerical fields

In [240…

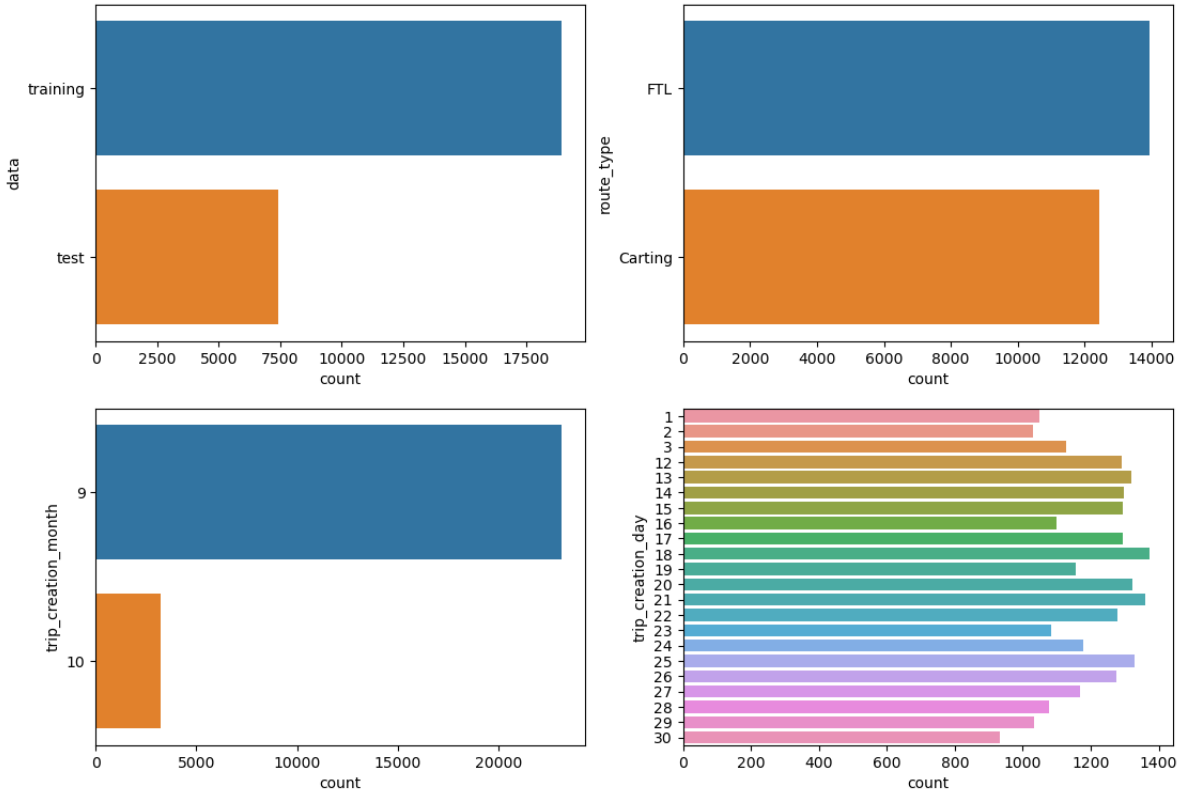
```
#Univariate analysis of numerical fields
num_cols = ['segment_actual_time_sum', 'segment_osrm_distance_sum', 'segment_osrm_
            'actual_time', 'osrm_time','osrm_distance']
fig, axis = plt.subplots(nrows=3, ncols=2, figsize=(10, 12))
index = 0
for row in range(3):
    for col in range(2):
        sns.histplot(data_feature[num_cols[index]], ax=axis[row, col], kde=True)
        index += 1
plt.show()
```



Univariate analysis of categorical fields

In [240…

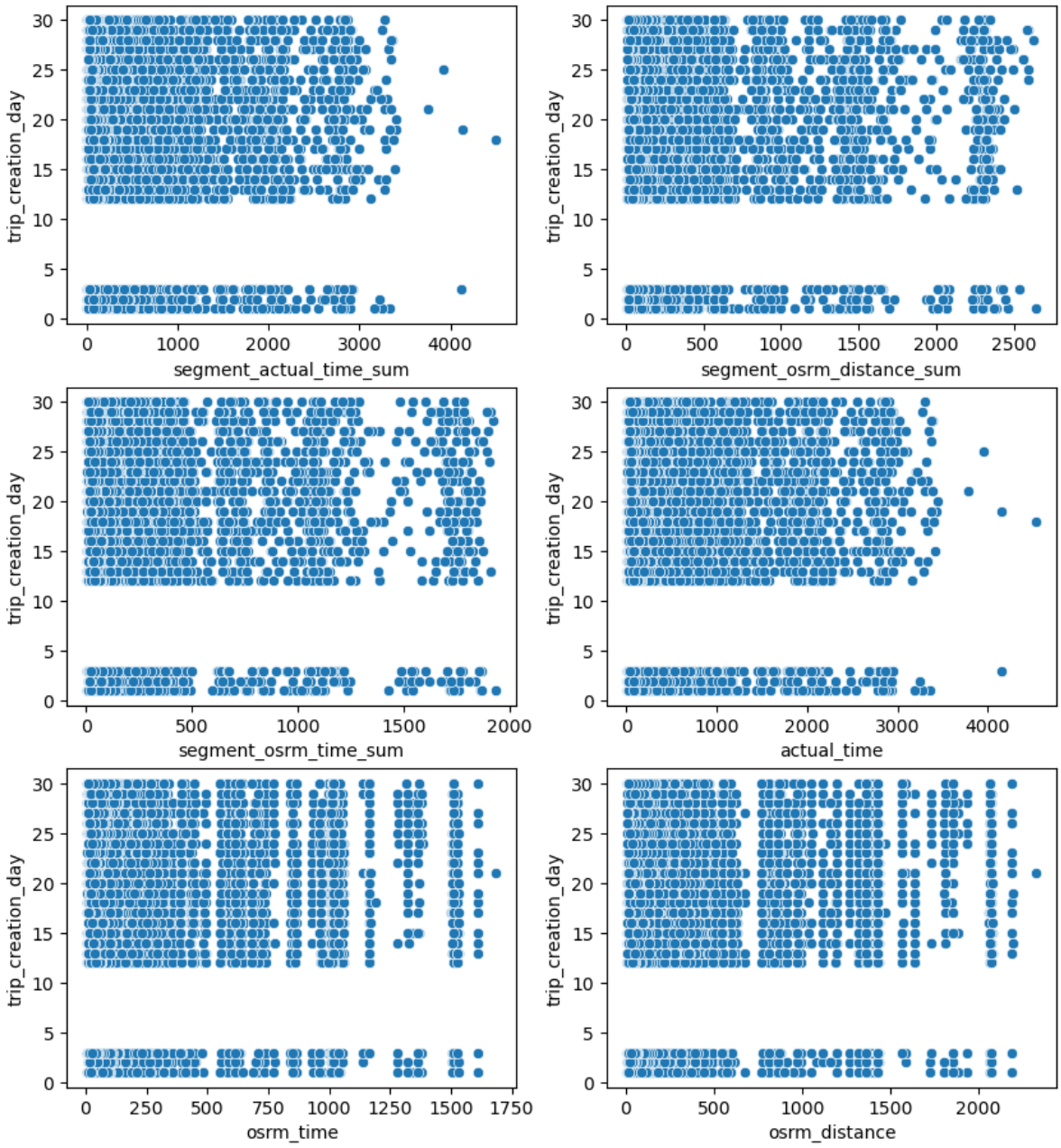
```
cat_columns= ['data', 'route_type', 'trip_creation_month', 'trip_creation_day']
fig, axis = plt.subplots(nrows=2, ncols=2, figsize=(13, 9))
index = 0
for row in range(2):
    for col in range(2):
        sns.countplot(data=data_feature, y=cat_columns[index], ax=axis[row, col])
        index += 1
plt.show()
```



Bivariate analysis of numerical fields

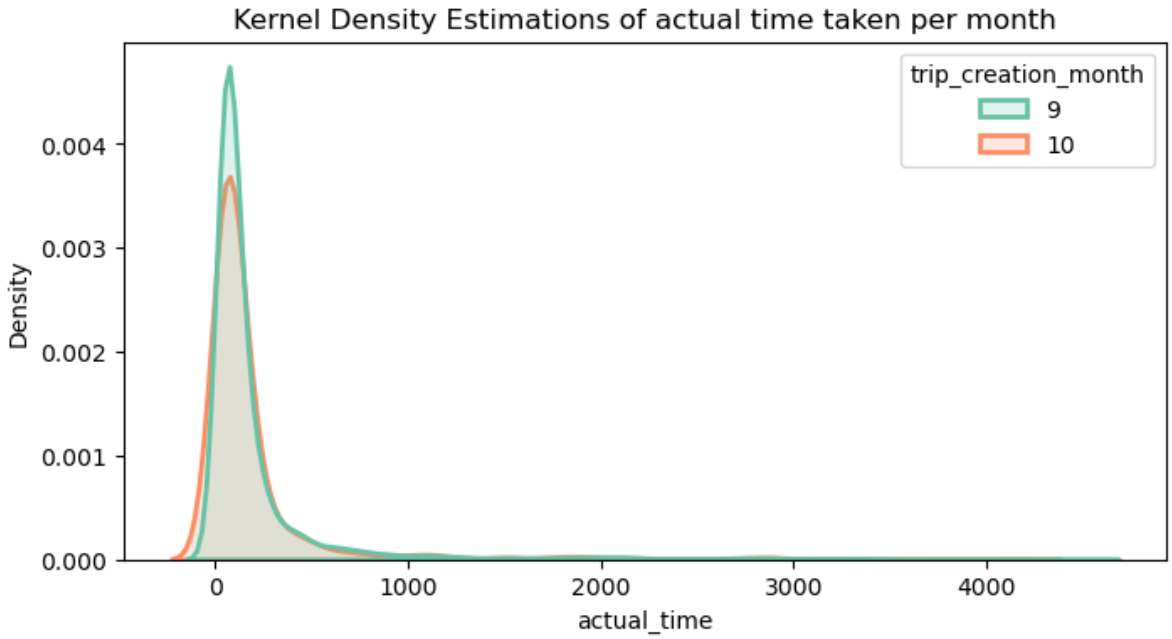
In [240…

```
num_columns= ['segment_actual_time_sum', 'segment_osrm_distance_sum', 'segment_osrm
            'actual_time', 'osrm_time','osrm_distance']
fig, axis = plt.subplots(nrows=3, ncols=2, figsize=(10, 11))

index = 0
for row in range(3):
    for col in range(2):
        sns.scatterplot(data=data_feature, x=num_columns[index], y='trip_creation_
        index += 1
plt.show()
```



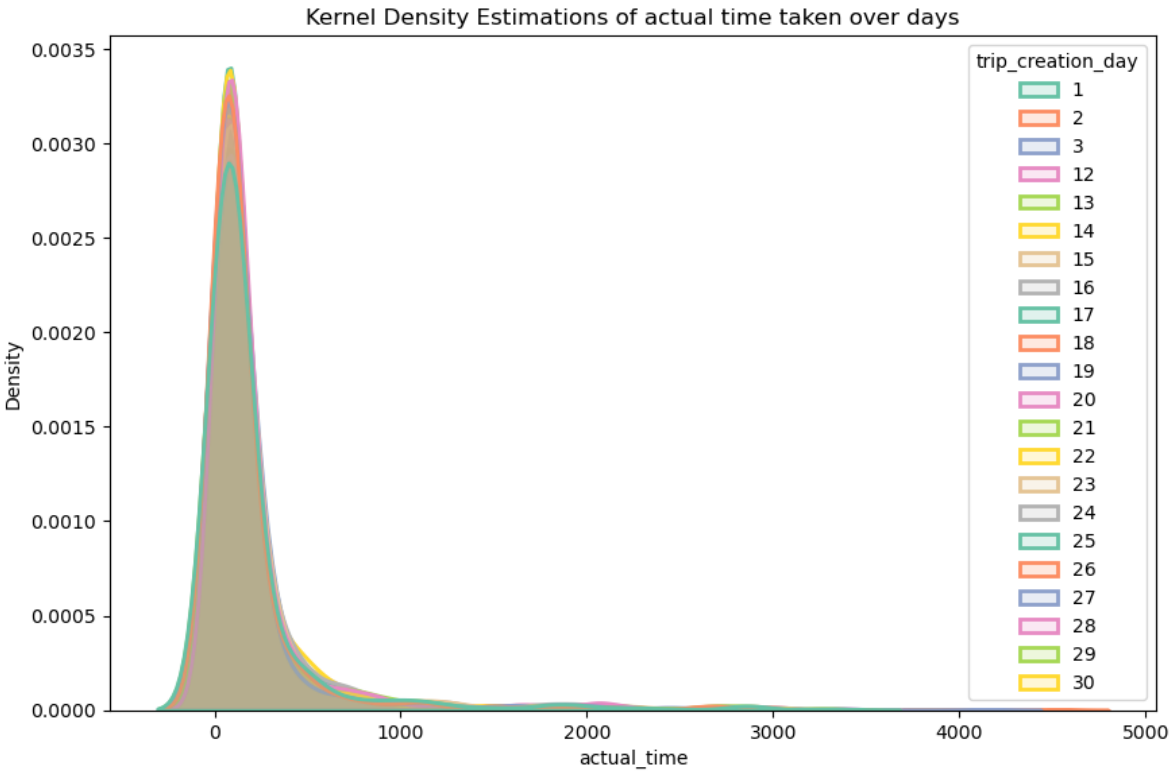
In [240…

```
plt.figure(figsize = (8,4))
sns.kdeplot(data=data_feature, x="actual_time", hue="trip_creation_month", fill=Tru
            palette='Set2', alpha=.2, linewidth=2)
plt.title("Kernel Density Estimations of actual time taken per month")
plt.show()
```



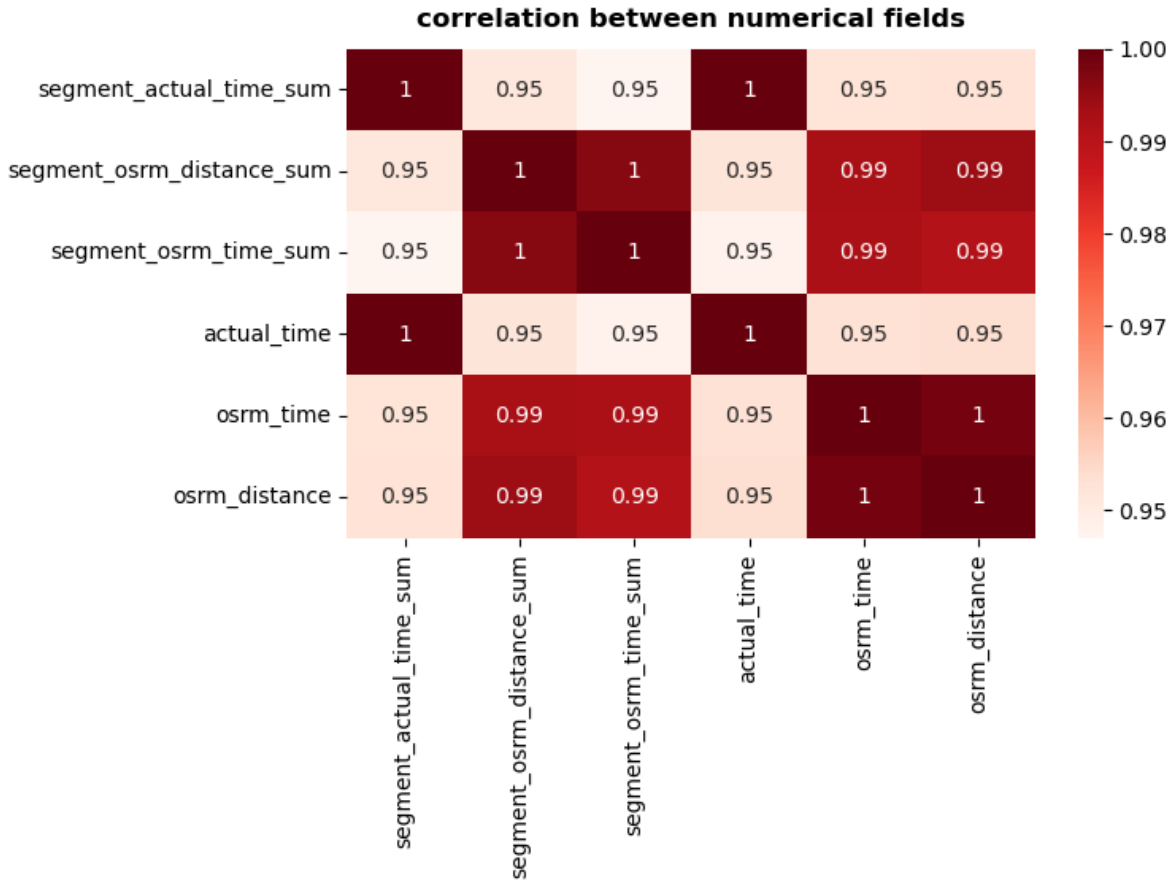
In [240…

```
plt.figure(figsize = (10,6.5))
sns.kdeplot(data=data_feature, x="actual_time", hue="trip_creation_day", fill=True,
            palette='Set2', alpha=.2, linewidth=2)
plt.title("Kernel Density Estimations of actual time taken over days")
plt.show()
```



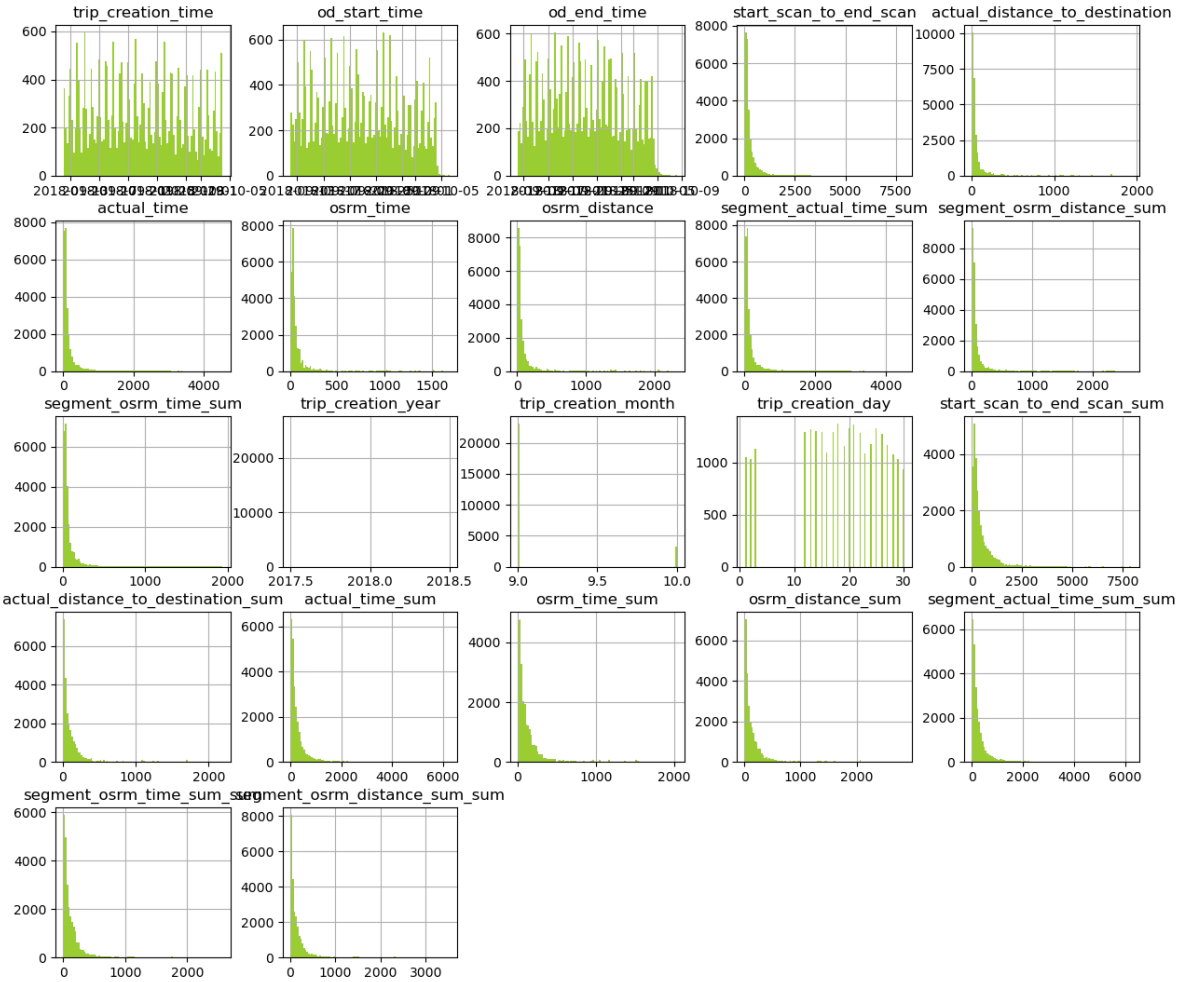
In [240…

```
plt.figure(figsize = (7,4))
data_feature_corr = data_feature[['segment_actual_time_sum', 'segment_osrm_distance
                        'segment_osrm_time_sum','actual_time','osrm_time','osrm_dis
sns.heatmap(data = data_feature_corr,annot = True , cmap = "Reds")
plt.title("correlation between numerical fields", loc = "center",pad= 10,fontweight
plt.show(block = False)
```



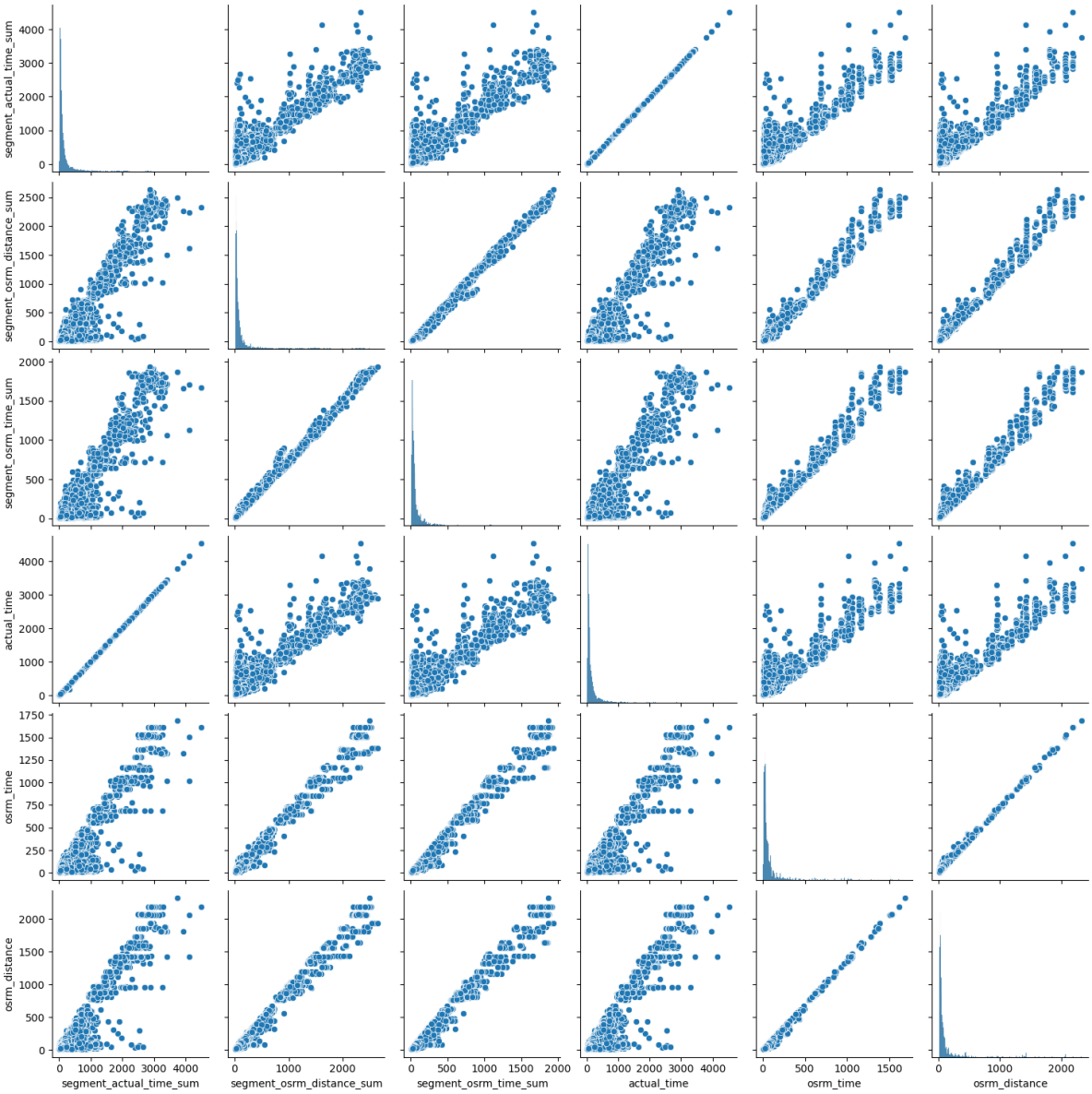
In [240…

```
data_feature.hist(bins=100,figsize=(15,13),color= "yellowgreen")
plt.show()
```



In [240…

```
sns.pairplot(data_feature[['segment_actual_time_sum', 'segment_osrm_distance_sum',
            'actual_time', 'osrm_time','osrm_distance']],palette = "set3")
plt.show()
```

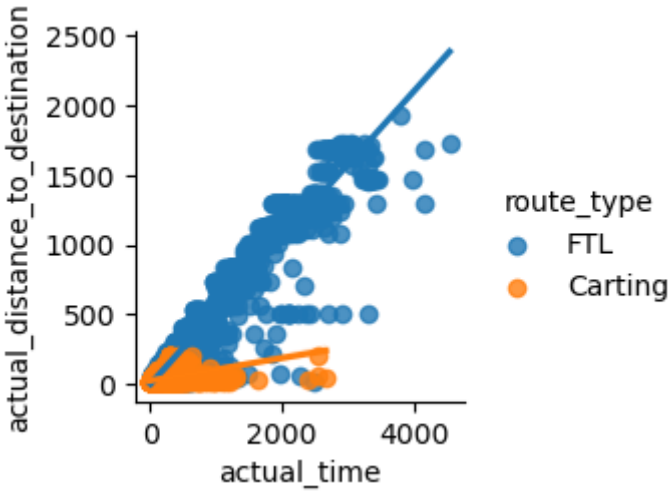


Exploratory Data Analysis (EDA)

Which route_type attained maximum speed ?

In [241…

```
pair = ("actual_time","actual_distance_to_destination")
sns.pairplot(data=data_feature,kind="reg",hue="route_type", x_vars=[pair[0]], y_va
plt.show()
```

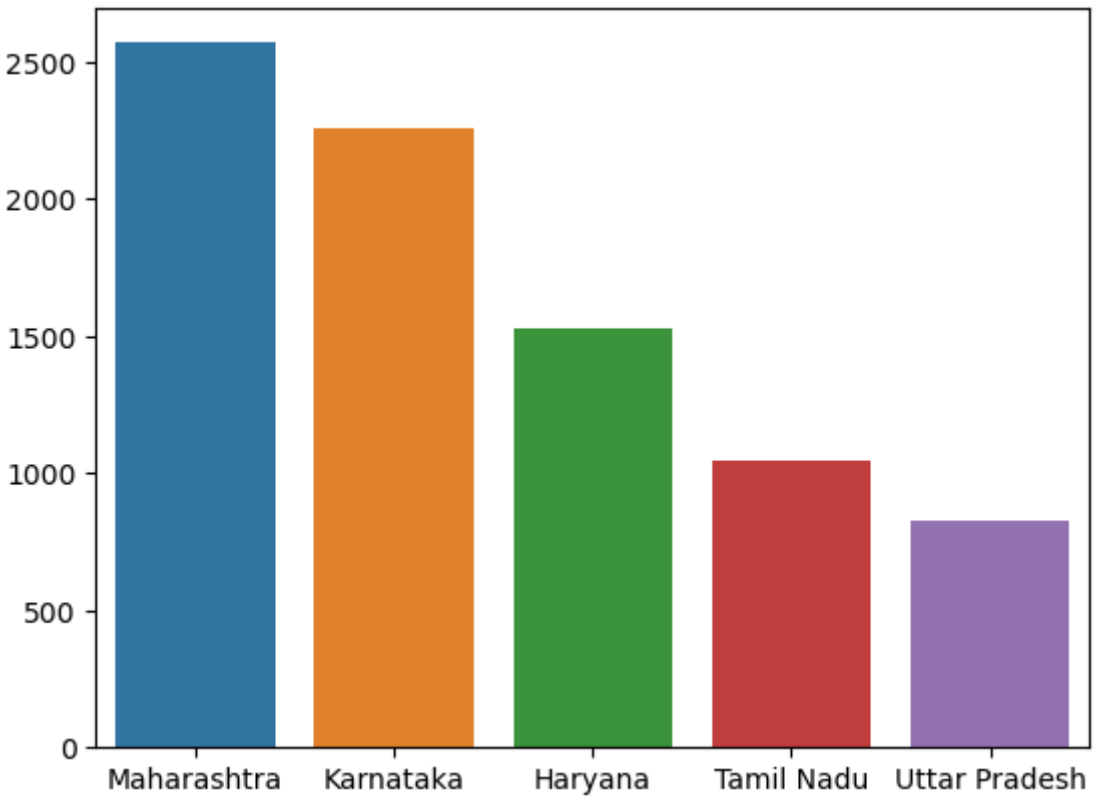


If ignored all last upper whisker values, it can be observed from the plot that speed attained
in route_type FTL is more than speed attained in route_type Carting

Check from where most orders are coming from (State, Corridor etc)

In [241…

```
sns.barplot(x=df_feature["destination_state"].value_counts()[:5].index,
            y=df_feature["destination_state"].value_counts()[:5].values)
plt.show()
```

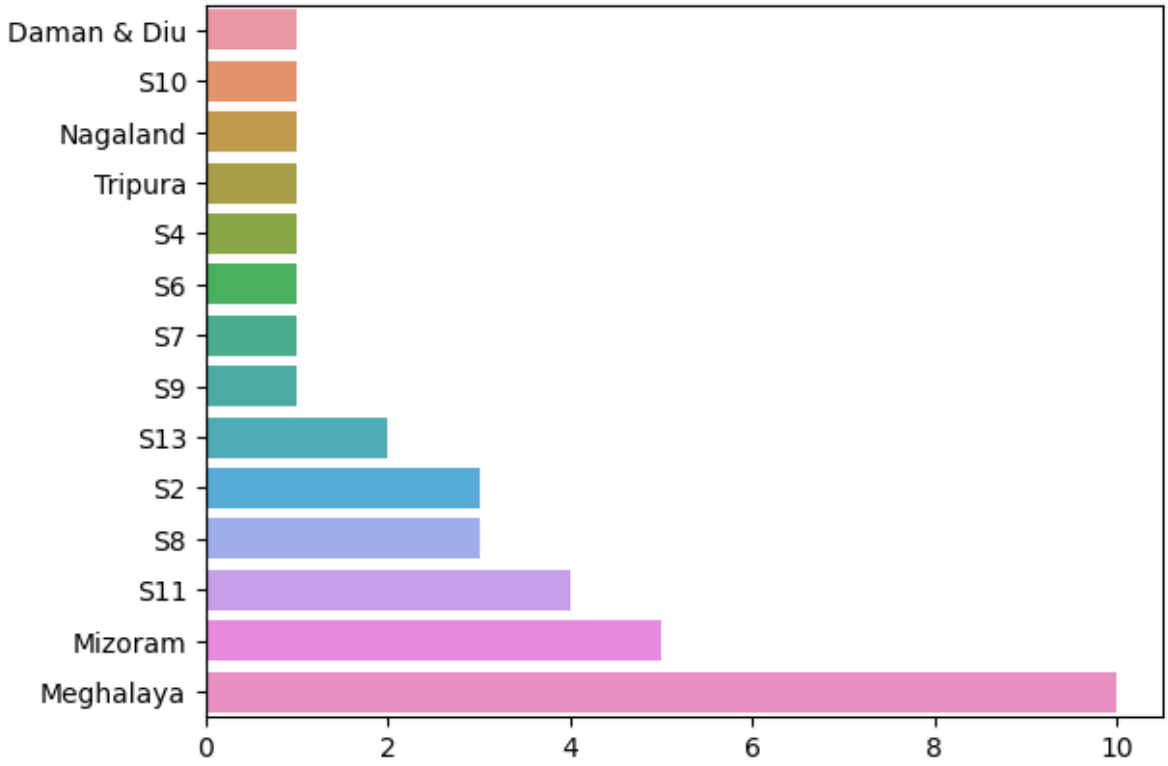


Most orders are coming from Maharashtra, followed by Karanataka.
Above plot shows top 5 states from where orders are coming most.

Check from where least orders are coming from (State, Corridor etc)

In [241…

```
sns.barplot(y=df_feature["destination_state"].value_counts()[-1:-15:-1].index,
            x=df_feature["destination_state"].value_counts()[-1:-15:-1].values)
plt.show()
```

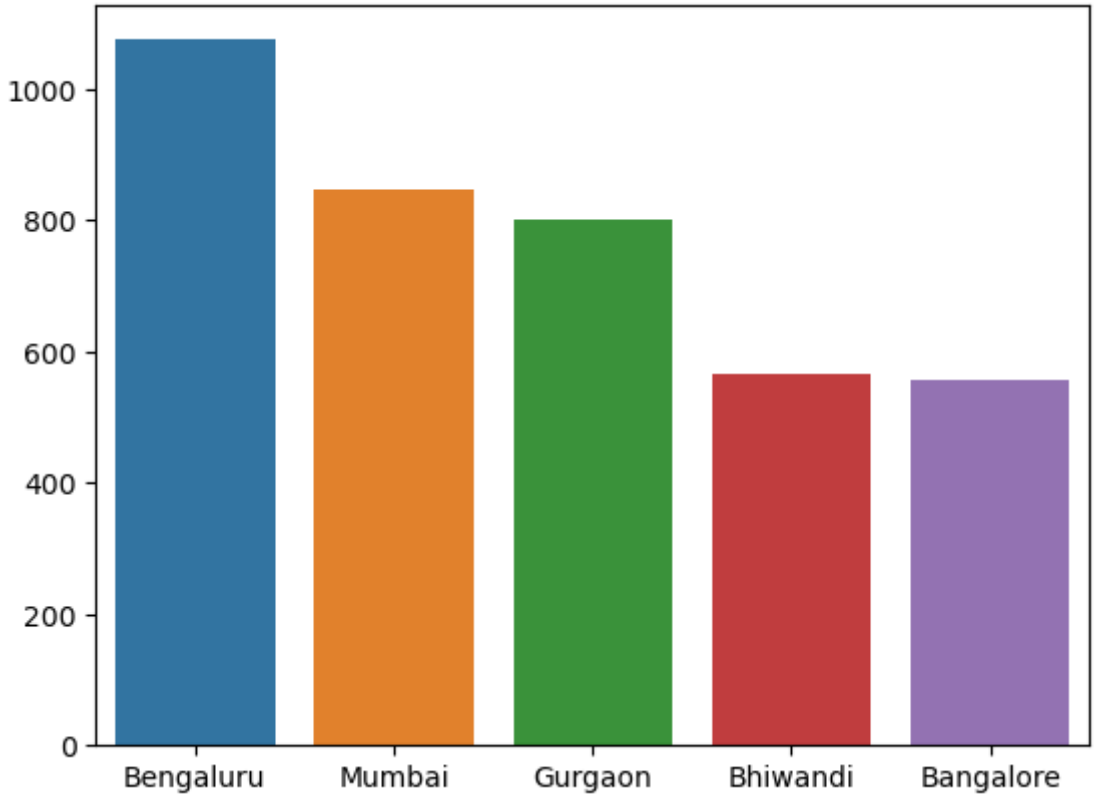


The above plot is showing top 15 states with least orders.
If observed, except Damana and Diu, all states belong to north-east region of the country.
North-East states are making least orders.

Check from where which city most orders are coming from ?

In [241…

```
sns.barplot(x=df_feature["destination_city"].value_counts()[:5].index,
            y=df_feature["destination_city"].value_counts()[:5].values)
plt.show()
```

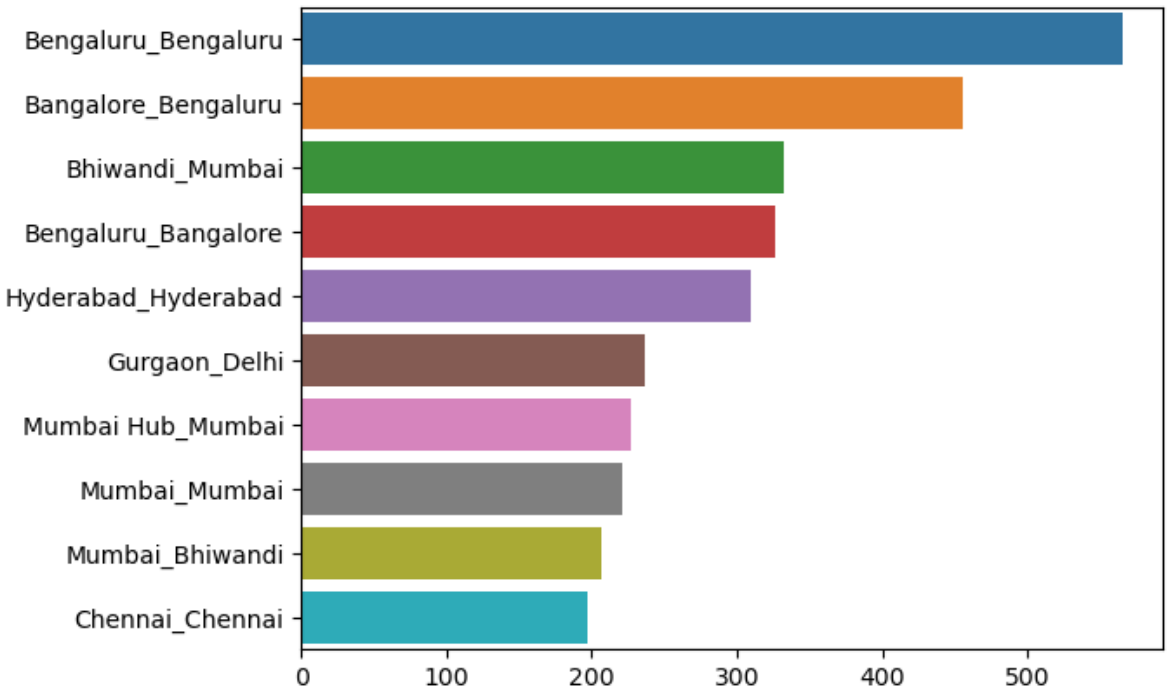


The most orders are coming from metro cities like Bengaluru, Mumbai and Gurgaon.
The urban people are ordering way more than people living in sub-urban or small cities.

Busiest corridor, avg distance between them, avg time taken

In [241…

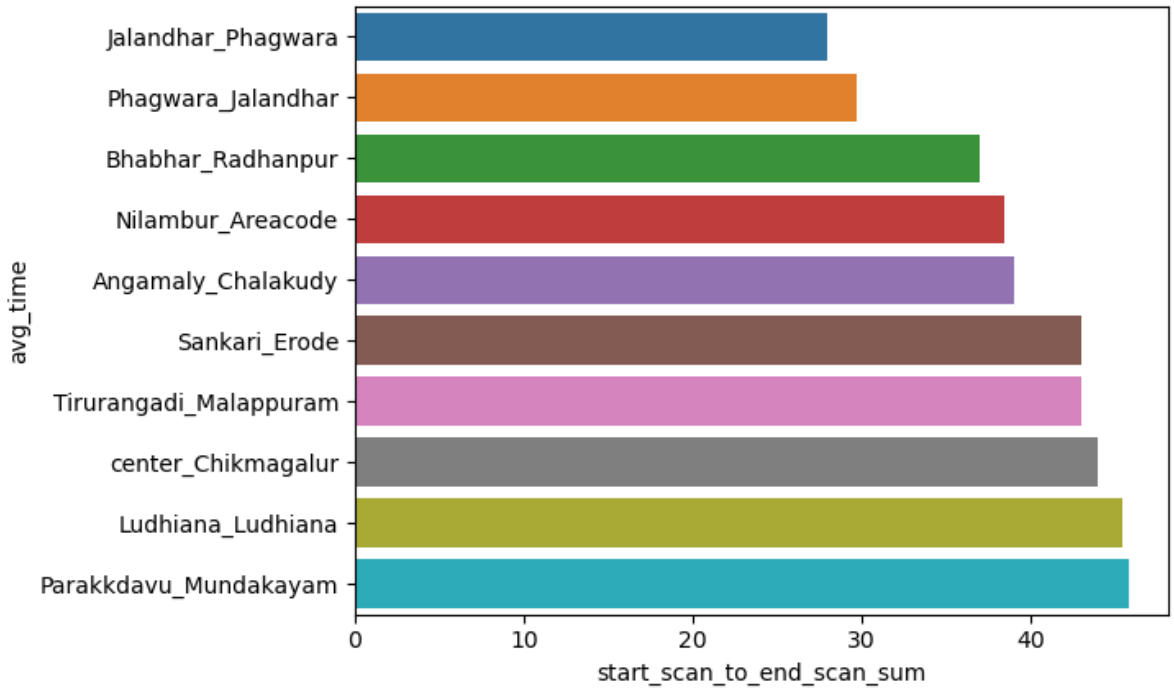
```
# busiest corridor
busiest = (df_feature["source_city"]+"_"+df_feature["destination_city"]).value_cou
sns.barplot(y = busiest.index, x = busiest.values)
plt.show()
```



The busiest corridor is Bangalore to Bangalore (within city). However, this corridor is shortest among many others.
The corridor Bhiwandi to Mumbai is also the busiest and longer among others.

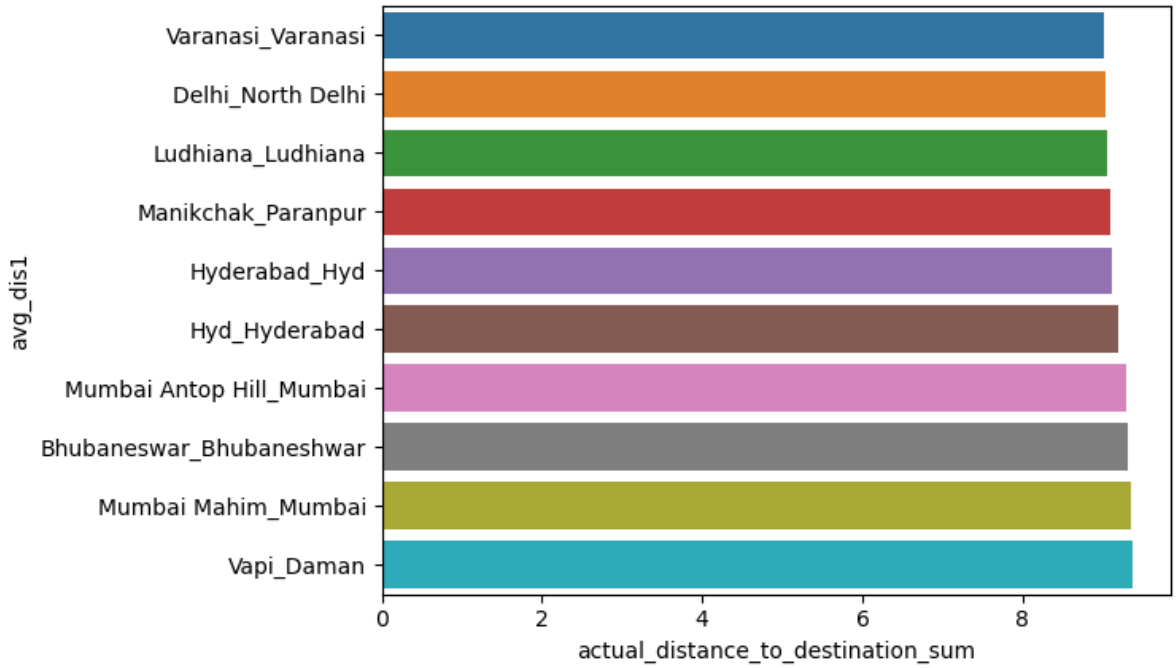
In [241…

```
# top 10 corridors having less average time taken
time_total = df_feature.groupby(["source_city","destination_city"]).agg(sum)
time_count = df_feature.groupby(["source_city","destination_city"]).count()
avg_time_taken = time_total/time_count
avg_time_taken_low = avg_time_taken.sort_values(by=['start_scan_to_end_scan_sum'],
                                                ascending=True)[['start_scan_to_en
avg_time_taken_low = avg_time_taken_low.reset_index()
avg_time_taken_low["avg_time"] = avg_time_taken_low["source_city"]+"_"+avg_time_tal

sns.barplot(y = avg_time_taken_low["avg_time"], x = avg_time_taken_low['start_scan_
plt.show()
```



In [241…

```
# top 10 corridors having manimum average distance
dis_total = df_feature.groupby(["source_city","destination_city"]).agg(sum)
dis_count = df_feature.groupby(["source_city","destination_city"]).count()
avg_dis = dis_total/dis_count
avg_dis = avg_dis.sort_values(by=['actual_distance_to_destination_sum'],
                                                ascending=True)[['actual_distance_
avg_dis = avg_dis.reset_index()
avg_dis["avg_dis1"] = avg_dis["source_city"]+"_"+avg_dis["destination_city"]

sns.barplot(y = avg_dis["avg_dis1"], x = avg_dis['actual_distance_to_destination_su
plt.show()
```

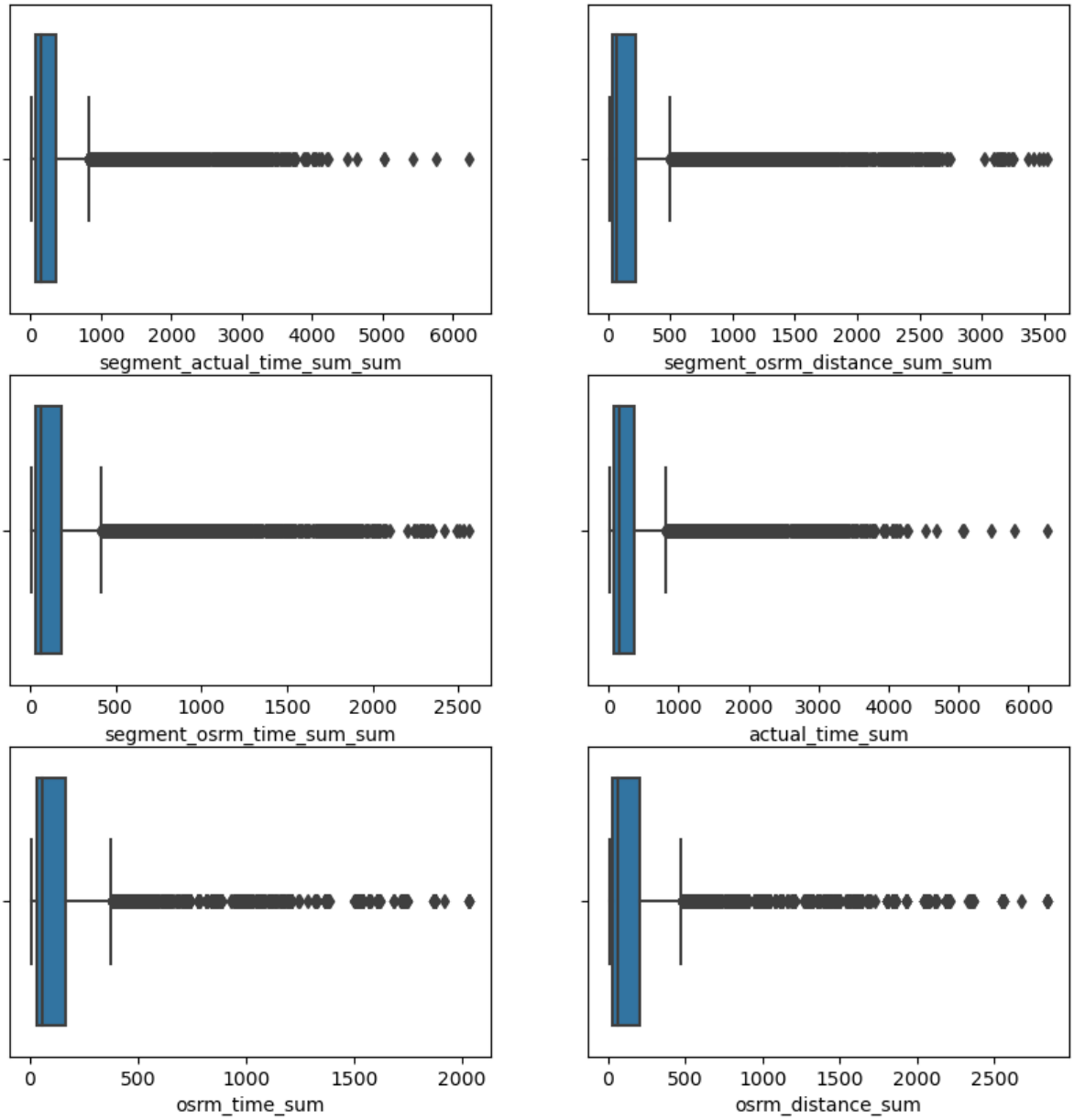


Outliers Analysis

Visual Analysis for outliers

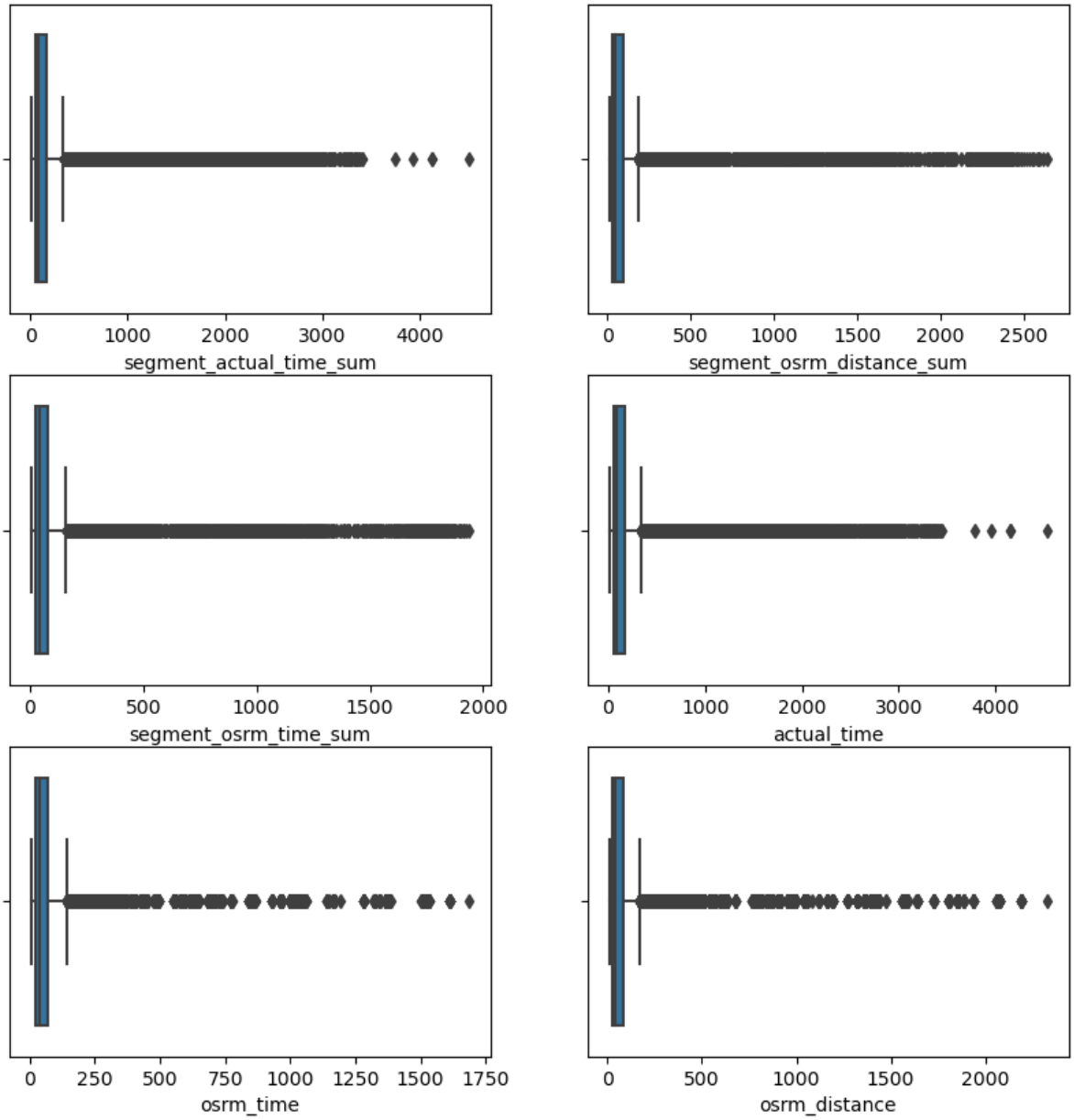
In [241…

```
# plotting box plots to detect outliers in the data grouped by trip_uuid
num_cols = ['segment_actual_time_sum_sum', 'segment_osrm_distance_sum_sum', 'segmer
            'actual_time_sum', 'osrm_time_sum','osrm_distance_sum']
fig, axis = plt.subplots(nrows=3, ncols=2, figsize=(10, 10))
index = 0
for row in range(3):
    for col in range(2):
        sns.boxplot(x=df_feature[num_cols[index]], ax=axis[row, col])
        index += 1
plt.show()
```



In [241…

```
# plotting box plots to detect outliers in the data grouped by trip_uuid.source and
num_cols = ['segment_actual_time_sum', 'segment_osrm_distance_sum', 'segment_osrm_t
            'actual_time', 'osrm_time','osrm_distance']
fig, axis = plt.subplots(nrows=3, ncols=2, figsize=(10, 10))
index = 0
for row in range(3):
    for col in range(2):
        sns.boxplot(x=data_feature[num_cols[index]], ax=axis[row, col])
        index += 1
plt.show()
```



In both the datasets, there are high number of outliers in all fields

Mathematically detect % of Outliers in various fields

Calculating IQR, upper and lower bound for numeric fields in "data_feature" dataset (grouped by trip id, source and destination id)

In [241…

```
# Calculating IQR, upper and lower bound for numeric fields in data_feature dataset

data_num_cols = ['segment_actual_time_sum', 'segment_osrm_distance_sum', 'segment_
            'actual_time', 'osrm_time','osrm_distance']

num_cols_outlier_upper_value = []
num_cols_outlier_upper_value_percentage = []
num_cols_outlier_lower_value = []
num_cols_outlier_lower_value_percentage = []

for col in data_num_cols:
    Q1_count = np.percentile(data_feature[col], 25,interpolation = 'midpoint')
    Q3_count = np.percentile(data_feature[col], 75,interpolation = 'midpoint')
    IQR_count = Q3_count - Q1_count
```

```
    upper_count=Q3_count+1.5*IQR_count
    lower_count=Q1_count-1.5*IQR_count

    if lower_count < 0:
        lower_count = 0

    upper_len = len(data_feature[data_feature[col]>=upper_count])
    lower_len = len(data_feature[data_feature[col]<=lower_count])
    dataset_len = len(data_feature)
    num_cols_outlier_upper_value.append(upper_len)
    num_cols_outlier_lower_value.append(lower_len)
    num_cols_outlier_upper_value_percentage.append(round(upper_len/dataset_len,2))
    num_cols_outlier_lower_value_percentage.append(round(lower_len/dataset_len,2))

outlier_data = pd.DataFrame({ 'Fields': num_cols,
                              'upper outliers': num_cols_outlier_upper_value,
                             'lower outliers': num_cols_outlier_lower_value,
                            'upper outlier %':num_cols_outlier_upper_value_percentage
                            'lower outlier %': num_cols_outlier_lower_value_percentage
outlier_data
```

Out[2419]:

	Fields	upper outliers	lower outliers	upper outlier %	lower outlier %
0	segment_actual_time_sum	3160	0	0.12	0.0
1	segment_osrm_distance_sum	3105	0	0.12	0.0
2	segment_osrm_time_sum	3167	0	0.12	0.0
3	actual_time	3152	0	0.12	0.0
4	osrm_time	2919	0	0.11	0.0
5	osrm_distance	3098	0	0.12	0.0

Calculating IQR, upper and lower bound for numeric fields in "df_feature" dataset (further grouped by trip id)

In [242…

```
# Calculating IQR, upper and lower bound for numeric fields in df_feature dataset
df_num_cols = ['segment_actual_time_sum_sum', 'segment_osrm_distance_sum_sum', 'seg
            'actual_time_sum', 'osrm_time_sum','osrm_distance_sum','start_scan_to_
            'actual_distance_to_destination_sum']

df_num_cols_outlier_upper_value = []
df_num_cols_outlier_upper_value_percentage = []
df_num_cols_outlier_lower_value = []
df_num_cols_outlier_lower_value_percentage = []

for col in df_num_cols:
    Q1 = np.percentile(df_feature[col], 25,interpolation = 'midpoint')
    Q3 = np.percentile(df_feature[col], 75,interpolation = 'midpoint')
    IQR = Q3 - Q1
    upper =Q3 +1.5*IQR
    lower=Q1-1.5*IQR

    if lower < 0:
        lower = 0

    upper_len = len(df_feature[df_feature[col]>=upper])
    lower_len = len(df_feature[df_feature[col]<=lower])
    dataset_len = len(df_feature)
    df_num_cols_outlier_upper_value.append(upper_len)
    df_num_cols_outlier_lower_value.append(lower_len)
    df_num_cols_outlier_upper_value_percentage.append(round(upper_len/dataset_len,
    df_num_cols_outlier_lower_value_percentage.append(round(lower_len/dataset_len,
```

```
outlier_df = pd.DataFrame({ 'Fields': df_num_cols,
                            'upper outliers': df_num_cols_outlier_upper_value,
                           'lower outliers': df_num_cols_outlier_lower_value,
                          'upper outlier %':df_num_cols_outlier_upper_value_percenta
                          'lower outlier %': df_num_cols_outlier_lower_value_percent
outlier_df
```

Out[2420]:

	Fields	upper outliers	lower outliers	upper outlier %	lower outlier %
0	segment_actual_time_sum_sum	1643	0	0.11	0.0
1	segment_osrm_distance_sum_sum	1548	0	0.10	0.0
2	segment_osrm_time_sum_sum	1494	0	0.10	0.0
3	actual_time_sum	1643	0	0.11	0.0
4	osrm_time_sum	1517	0	0.10	0.0
5	osrm_distance_sum	1524	0	0.10	0.0
6	start_scan_to_end_scan_sum	1267	0	0.09	0.0
7	actual_distance_to_destination_sum	1449	0	0.10	0.0

Treating outliers by replacing outliers with upper bound and lower bound

In [242…

```
data_num_cols = ['segment_actual_time_sum', 'segment_osrm_distance_sum', 'segment_
            'actual_time', 'osrm_time','osrm_distance']

for col in data_num_cols:
    Q1 = data_feature[col].quantile(0.25)
    Q3 = data_feature[col].quantile(0.75)
    IQR = Q3 - Q1
    S = 1.5*IQR
    LB = Q1 - S
    UB = Q3 + S
    data_feature.loc[data_feature[col] > UB,col] = UB
    data_feature.loc[data_feature[col] < LB,col] = LB
```

In [242…

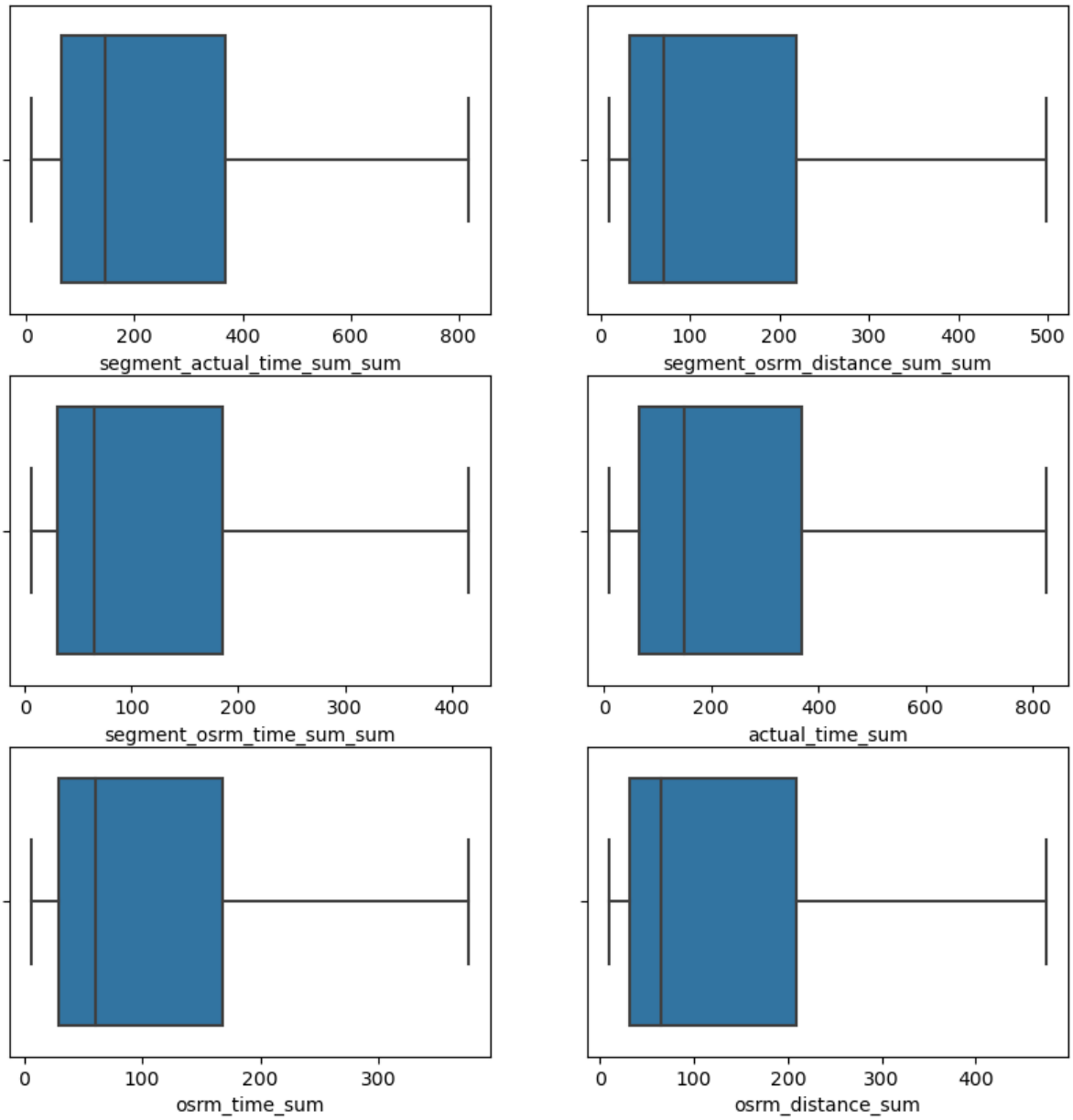
```
df_num_cols = ['segment_actual_time_sum_sum', 'segment_osrm_distance_sum_sum', 'seg
            'actual_time_sum', 'osrm_time_sum','osrm_distance_sum','start_scan_to_
            'actual_distance_to_destination_sum']

for col in df_num_cols:
    Q1 = df_feature[col].quantile(0.25)
    Q3 = df_feature[col].quantile(0.75)
    IQR = Q3 - Q1
    S = 1.5*IQR
    LB = Q1 - S
    UB = Q3 + S
    df_feature.loc[df_feature[col] > UB,col] = UB
    df_feature.loc[df_feature[col] < LB,col] = LB
```

Checking for outliers again

In [242…

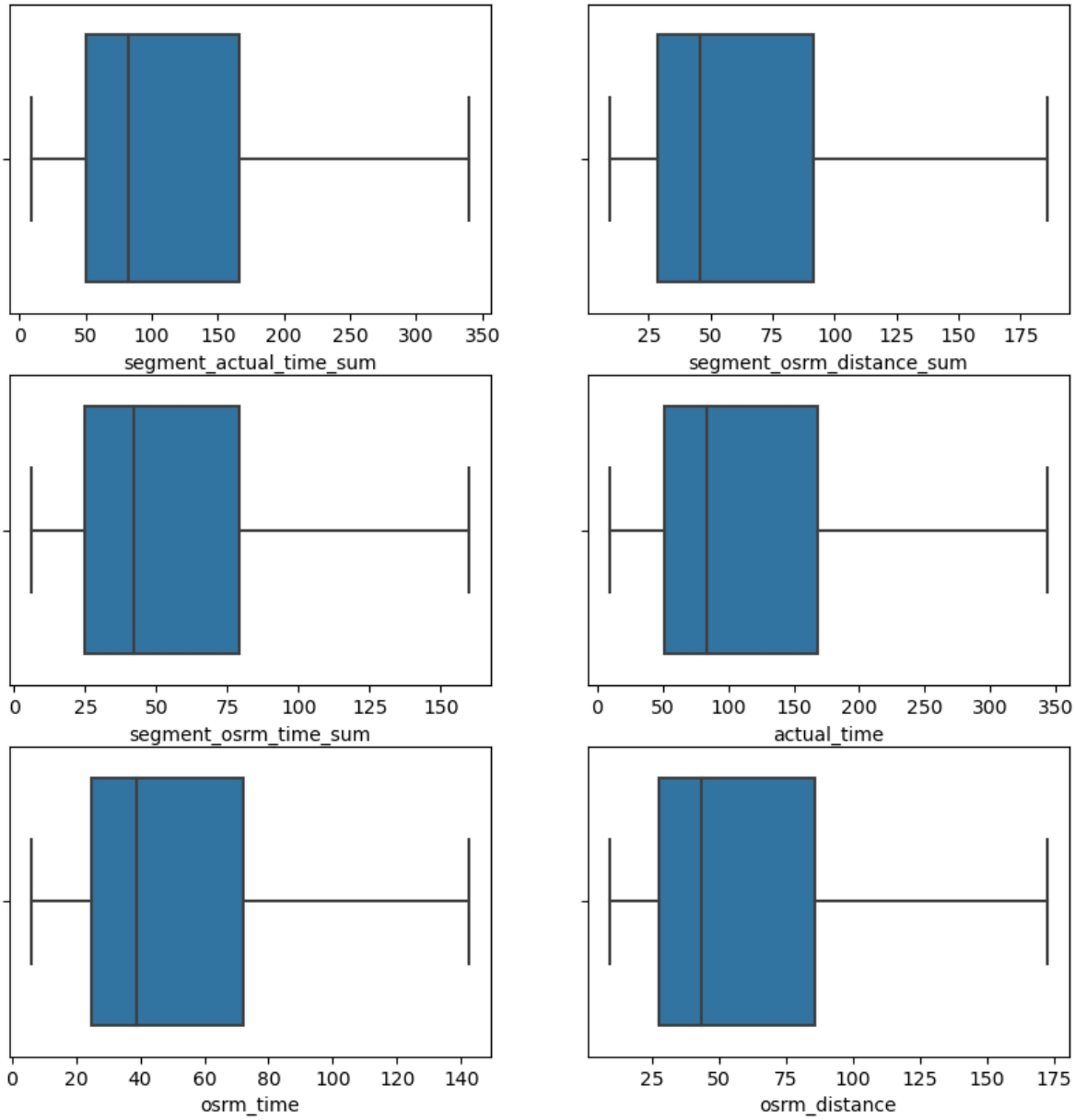
```
# plotting box plots to detect outliers in the data grouped by trip_uuid
num_cols = ['segment_actual_time_sum_sum', 'segment_osrm_distance_sum_sum', 'segmen
            'actual_time_sum', 'osrm_time_sum','osrm_distance_sum']
```

```
fig, axis = plt.subplots(nrows=3, ncols=2, figsize=(10, 10))
index = 0
for row in range(3):
    for col in range(2):
        sns.boxplot(x=df_feature[num_cols[index]], ax=axis[row, col])
        index += 1
plt.show()
```



In [242…

```
# plotting box plots to detect outliers in the data grouped by trip_uuid.source and
num_cols = ['segment_actual_time_sum', 'segment_osrm_distance_sum', 'segment_osrm_
            'actual_time', 'osrm_time','osrm_distance']
fig, axis = plt.subplots(nrows=3, ncols=2, figsize=(10, 10))
index = 0
for row in range(3):
    for col in range(2):
        sns.boxplot(x=data_feature[num_cols[index]], ax=axis[row, col])
        index += 1
plt.show()
```



In-depth analysis and feature engineering

1. Calculate the time taken between od_start_time and od_end_time and keep it as a feature. Drop the original columns, if required

In [242…

```
# creating new feature; difference in start and end time
data_feature["time_diff_start_end"]=(data_feature["od_end_time"]-data_feature["od_
display(data_feature[["od_start_time","od_end_time","time_diff_start_end","start_s
```

	od_start_time	od_end_time	time_diff_start_end	start_scan_to_end_scan
0	2018-09-12 16:39:46.858469	2018-09-13 13:40:23.123744	1260.604421	1260.0
1	2018-09-12 00:00:16.535741	2018-09-12 16:39:46.858469	999.505379	999.0
2	2018-09-12 02:03:09.655591	2018-09-12 03:01:59.598855	58.832388	58.0
3	2018-09-12 00:00:22.886430	2018-09-12 02:03:09.655591	122.779486	122.0
4	2018-09-14 03:40:17.106733	2018-09-14 17:34:55.442454	834.638929	834.0
5	2018-09-12 00:00:33.691250	2018-09-14 03:40:17.106733	3099.723591	3099.0
6	2018-09-12 00:01:00.113710	2018-09-12 01:41:29.809822	100.494935	100.0
7	2018-09-12 00:02:09.740725	2018-09-12 02:34:10.515593	152.012914	152.0
8	2018-09-12 03:54:43.114421	2018-09-12 12:00:30.683231	485.792814	485.0
9	2018-09-12 02:34:10.515593	2018-09-12 03:54:43.114421	80.543314	80.0

Hypothesis testing

Basic approach for all the tests

1. Step-1 : Set up Null and alternate Hypothesis
2. Step-2 : Checking for basic assumpitons for the hypothesis
    A. whether data is continuous.
    B. nature of sample data
    C. distribution check using QQ plot
    D. homogeneity of variance (i.e., the variability of the data in each group is similar): It will be checked by levene's test
    E. The distribution is approximately normal: validate by shapiro wilk test
    F. If necessary
3. Step-3 : Define Test statistics; Distribution of Test statistics under H0.
4. Step-4: Decide the kind of test:
    A. Perform Two tailed t-test
    B. ANNOVA test
    C. Chi-square test
5. Step-5: Compute the p-value and fix value of alpha.
6. Step-6: Compare p-value and alpha.** Based on p-value, accept or reject H0.
    A. p-val > alpha : Accept H0
    B. p-val < alpha : Reject H0

In [242…

```
#Setting up a function to return result on the basis of the significance value(0.05
def result(p_value):
```

```
  if p_value < 0.05:
    print("Reject Null Hypothesis")
  else:
    print("Failed to Reject Null Hypothesis")
```

2. Compare the difference between "time_diff_start_end" and "start_scan_to_end_scan". Do hypothesis testing/ Visual analysis to check.

APPROACH

Step-1 : Hypothesis formulation

1. Null Hypothesis (H0) - mean of "time_diff_start_end" and "start_scan_to_end_scan" are same.
2. Alternate Hypothesis (HA) - mean of "time_diff_start_end" and "start_scan_to_end_scan" are not same.

Step-2 : Checking for basic assumpitons for the hypothesis

1. Distribution check using QQ Plot
2. Homogeneity of Variances using Lavene's test
3. Check normal distributions using shapiro wilk test

Step-3 : Define Test statistics; Distribution of T under H0.

1. Observe test statistic while performing a T-Test follows Tdistribution.

Step-4: Decide the kind of test.

1. Perform Two tailed t-test

Step-5: Compute the p-value and fix value of alpha.

1. compute t-test value using the ttest function using scipy.stats. set *alpha to be 0.05*

Step-6: Compare p-value and alpha.

1. Based on p-value, accept or reject H0.
   A. p-val > alpha : Accept H0
   B. p-val < alpha : Reject H0

Homogeneity of Variances using Lavene's test

In [242…
```
# Checking Homogeneity of Variances using Lavene's test
# Null hypothesis(Ho) : Samples have similar variances
# Alternate Hypothesis(Ha) : samples have different variances
levene_stat, p_value =stats.levene(data_feature["time_diff_start_end"] ,data_featu
print(levene_stat, p_value)
result(p_value)
```

```
3.181727460990207e-08 0.9998576726881699
Failed to Reject Null Hypothesis
```

Check normal distributions using shapiro wilk test

In [242…

```
# checking distribution of normality using shapiro wilk test
# Null hypothesis(Ho) : Sample is from the normal distributions
# Alternate Hypothesis(Ha): Sample is not from the normal distributions.
shapiro_stat, p_value =stats.shapiro(data_feature["time_diff_start_end"])
print(shapiro_stat, p_value)
result(p_value)
shapiro_stat, p_value =stats.shapiro(data_feature["start_scan_to_end_scan"])
print(shapiro_stat, p_value)
result(p_value)
```

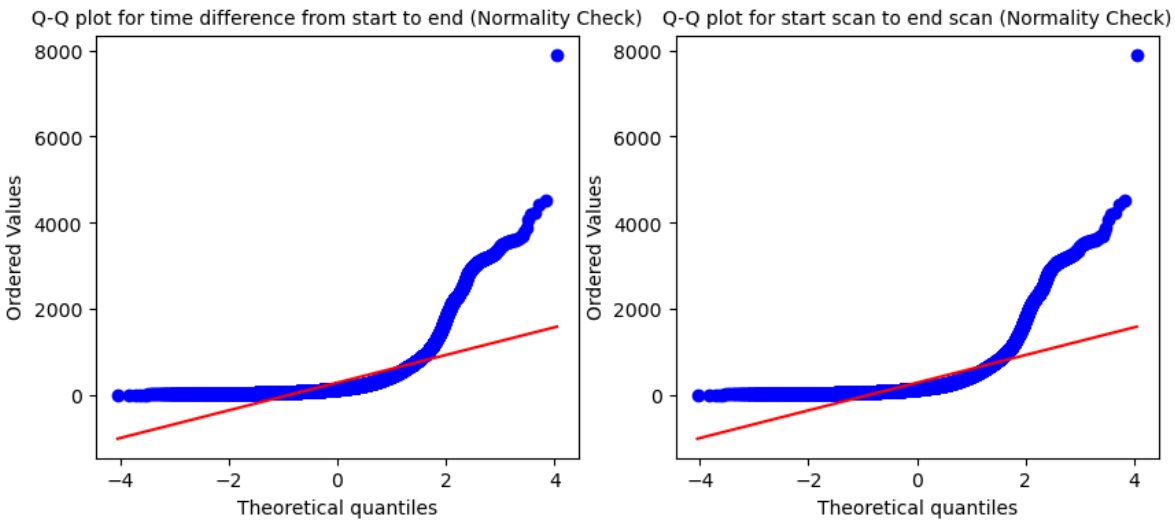
```
0.5302022695541382 0.0
Reject Null Hypothesis
0.5302027463912964 0.0
Reject Null Hypothesis
```

As null hypothesis has been rejected, this means that data samples does not follow normal distribution.
Now, verify the above result using Q-Q plots.

Distribution check using QQ plot

In [242…

```
#Construct QQ plots
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
stats.probplot(data_feature["time_diff_start_end"], plot= plt, dist="norm")
plt.title('Q-Q plot for time difference from start to end (Normality Check)', font
plt.subplot(1,2,2)
stats.probplot(data_feature["start_scan_to_end_scan"], plot= plt, dist="norm")
plt.title('Q-Q plot for start scan to end scan (Normality Check)', fontsize=10)
plt.show()
```

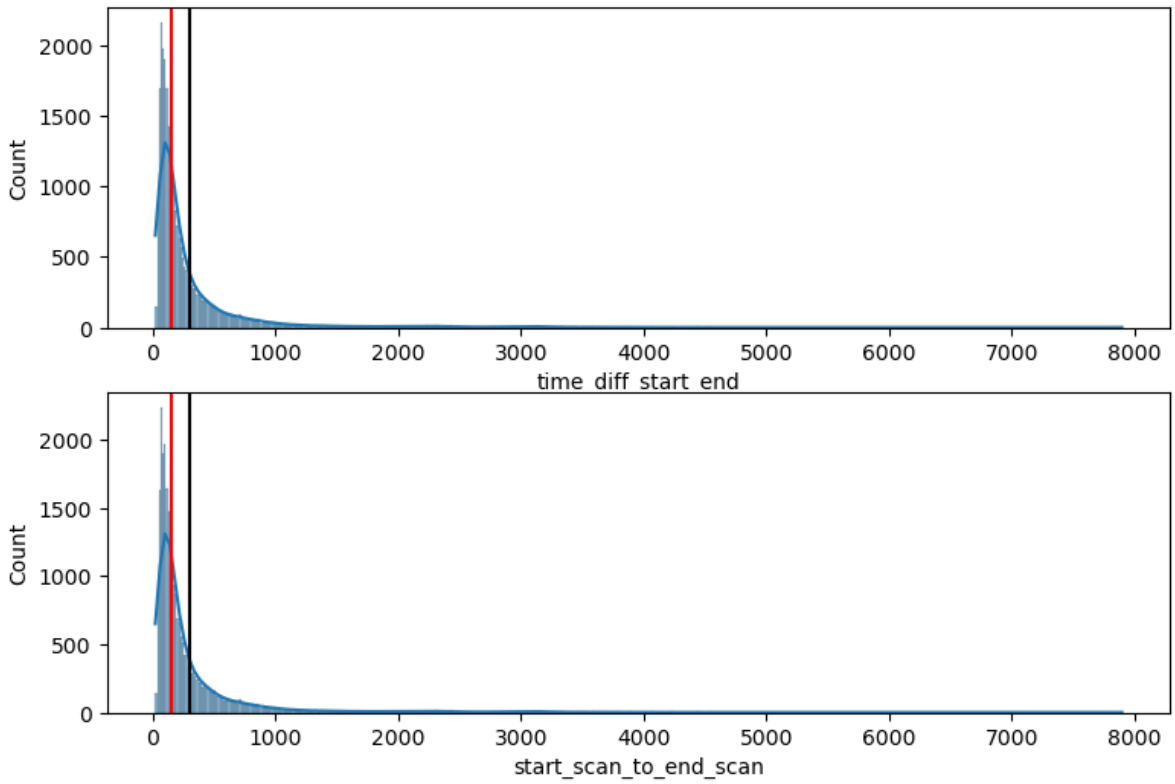


From above tests, it can be concluded that both samples have similar variances.
but, both samples are not actually following normal distribution.
Do visual testing to check which distribution these samples are following

visual testing

Checking distribution of samples

In [243…
```
plt.figure(figsize=(9,6))
plt.subplot(2,1,1)
sns.histplot(x=data_feature["time_diff_start_end"],kde=True)
plt.axvline(np.mean(data_feature["time_diff_start_end"]), c="k");
plt.axvline(np.quantile(data_feature["time_diff_start_end"],0.5), c="r")

plt.subplot(2,1,2)
sns.histplot(x=data_feature["start_scan_to_end_scan"],kde=True)
plt.axvline(np.mean(data_feature["start_scan_to_end_scan"]), c="k");
plt.axvline(np.quantile(data_feature["start_scan_to_end_scan"],0.5), c="r")
plt.show()
```



from above plots, it can be inferred that distribution is right tailed and highly skewed towards right.
verify this mathmematically by using function skew

Checking Skewness

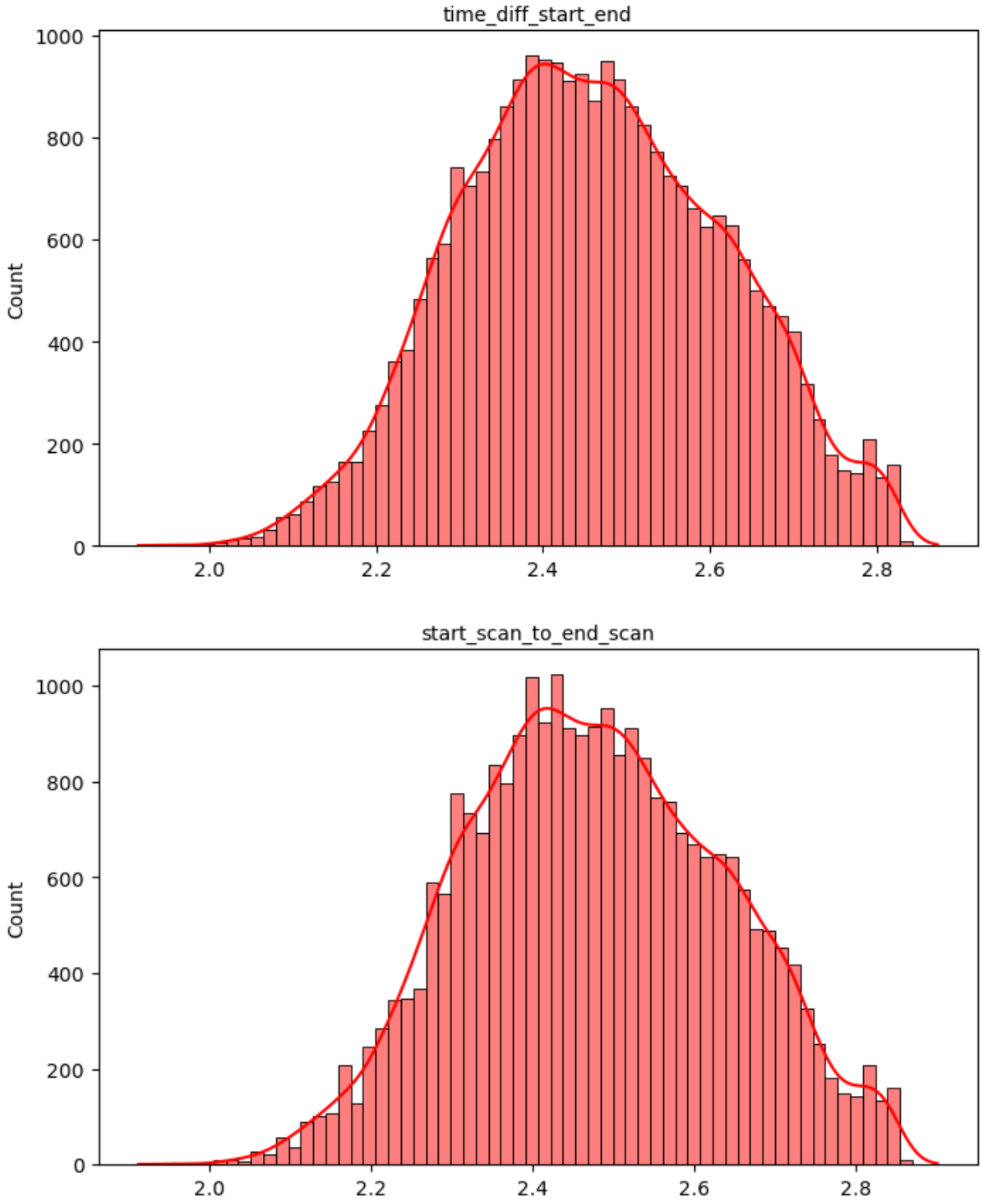
In [243…
```
(skew(data_feature["time_diff_start_end"]),skew(data_feature["start_scan_to_end_sc
```

Out[2431]: (4.213895288392638, 4.213868849477338)

As the value of skew is positive and more, it can be inferred that the distribution is highly
skewed towards right.
Or it is right tailed.
As observed from graphs, samples does not follow normal distribution,
so taking into consideration, apply BOXCOX transform to get normally distributed samples

In [243…
```
# BOX-COX Transform
```

```
time_diff= stats.boxcox(data_feature["time_diff_start_end"])[0]
scan_diff= stats.boxcox(data_feature["start_scan_to_end_scan"])[0]

fig, axis = plt.subplots(nrows=2, figsize=(8,9))
fig.subplots_adjust(top=1)
sns.histplot(data= time_diff,kde= True,ax= axis[0], color = "red")
axis[0].set_title("time_diff_start_end", pad=5, fontsize=10)
sns.histplot(data= scan_diff,kde= True,ax= axis[1], color = "red")
axis[1].set_title("start_scan_to_end_scan", pad=5, fontsize=10)
plt.show()
```



Now, the distribution is somewhat like normal distribution. Plot QQ plots again to verify

In [243…

```
# checking distribution of normality using shapiro wilk test
# Null hypothesis(Ho) : Sample is from the normal distributions
# Alternate Hypothesis(Ha): Sample is not from the normal distributions.
```

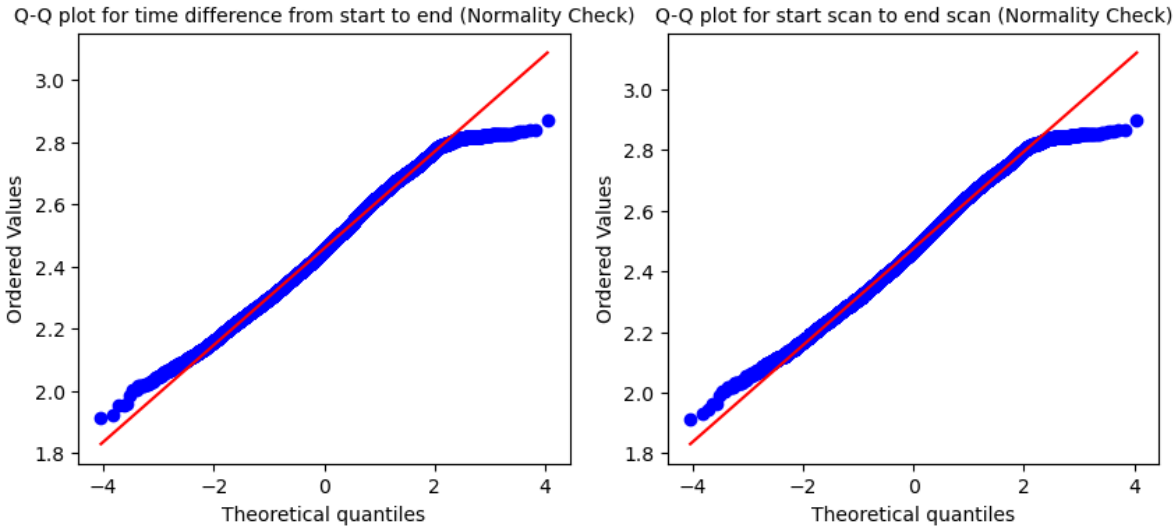
```
shapiro_stat, p_value =stats.shapiro(time_diff)
print(shapiro_stat, p_value)
result(p_value)
shapiro_stat, p_value =stats.shapiro(scan_diff)
print(shapiro_stat, p_value)
result(p_value)
```

```
0.9947192072868347 2.563482570939895e-29
Reject Null Hypothesis
0.9947843551635742 3.758779393635683e-29
Reject Null Hypothesis
```

Even after boxcox transform, shapiro test was rejecting null hypothesis.
It is to noted here that when your sample size is large, even the smallest deviation
from normality will lead to a rejected null.
Therefore, there is need to check normality visually by using QQ plots.

In [243…

```
#Construct QQ plots
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
stats.probplot(time_diff, plot= plt, dist="norm")
plt.title('Q-Q plot for time difference from start to end (Normality Check)', font
plt.subplot(1,2,2)
stats.probplot(scan_diff, plot= plt, dist="norm")
plt.title('Q-Q plot for start scan to end scan (Normality Check)', fontsize=10)
plt.show()
```



the samples are now following normal distributions, so taking into consideration that
t-test is tolarent towards samples beings not normaly distributed, apply 2-sample t_stat test.

performing t-test on samples

In [243…

```
# 2-sample t_stat test
t_stat, p_value = stats.ttest_ind(time_diff,scan_diff)
print("t_stat :",t_stat,"p_value :",p_value )
result(p_value)
```

```
t_stat : -12.336848483356105 p_value : 6.409344743236715e-35
Reject Null Hypothesis
```

```
    The above test reject null hypothesis. It concludes that mean
of "time_diff_start_end"
    and "start_scan_to_end_scan" are not same.
```

Performing Correlation test on samples

In [243…

```
# H0: No relation between time_diff and scan_diff
# Ha: relation between time_diff and scan_diff
pe_stat,p_value = pearsonr(x = time_diff, y = scan_diff )
print(pe_stat,p_value)
result(p_value)
```

```
0.9999891319836076 0.0
Reject Null Hypothesis
```

```
Correlation test tells whether there is relation between those
fields or not. They don't convey
the nature of relation.
There is relation between time difference and scan difference
```

3. Do hypothesis testing/ visual analysis between actual_time aggregated value and OSRM time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

APPROACH

Step-1 : Hypothesis formulation

1. Null Hypothesis (H0) - mean of "actual_time aggregated value" and "OSRM time aggregated value" are same.
2. Alternate Hypothesis (HA) - mean of "actual_time aggregated value" and "OSRM time aggregated value" are not same.

Step-2 : Checking for basic assumpitons for the hypothesis

1. Distribution check using QQ Plot
2. Homogeneity of Variances using Lavene's test
3. Check normal distributions using shapiro wilk test

Step-3 : Define Test statistics; Distribution of T under H0.

1. Observe test statistic while performing a T-Test follows Tdistribution.

Step-4: Decide the kind of test.

1. Perform Two tailed t-test

Step-5: Compute the p-value and fix value of alpha.

1. compute t-test value using the ttest function using scipy.stats. set *alpha to be 0.05*

Step-6: Compare p-value and alpha.

1. Based on p-value, accept or reject H0.
   A. p-val > alpha : Accept H0
   B. p-val < alpha : Reject H0

Homogeneity of Variances using Lavene's test

In [243…

```
# Checking Homogeneity of Variances using Lavene's test
# Null hypothesis(Ho) : Samples have similar variances
# Alternate Hypothesis(Ha) : samples have different variances
levene_stat, p_value =stats.levene(df_feature["actual_time_sum"] ,df_feature["osrm_
print(levene_stat, p_value)
result(p_value)
```

```
3152.9754634742044 0.0
Reject Null Hypothesis
```

Check normal distributions using shapiro wilk test

In [243…

```
# checking distribution of normality using shapiro wilk test
# Null hypothesis(Ho) : Sample is from the normal distributions
# Alternate Hypothesis(Ha): Sample is not from the normal distributions.
shapiro_stat, p_value =stats.shapiro(df_feature["actual_time_sum"])
print(shapiro_stat, p_value)
result(p_value)
shapiro_stat, p_value =stats.shapiro(df_feature["osrm_time_sum"])
print(shapiro_stat, p_value)
result(p_value)
```

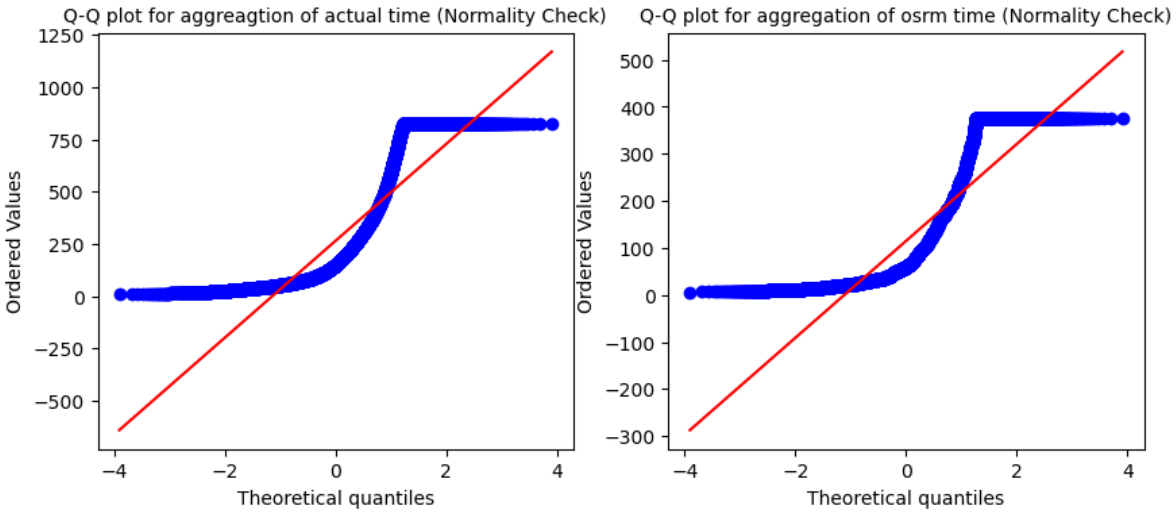
```
0.7887564897537231 0.0
Reject Null Hypothesis
0.7823594808578491 0.0
Reject Null Hypothesis
```

In levene's test and shapiro wilk test, null hypothesis has been rejected, this means that
data samples does not follow normal distribution and samples do not have similar variances
Now, verify the above result using Q-Q plots.

Distribution check using QQ plot

In [243…

```
#Construct QQ plots
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
stats.probplot(df_feature["actual_time_sum"], plot= plt, dist="norm")
plt.title('Q-Q plot for aggreagtion of actual time (Normality Check)', fontsize=10
plt.subplot(1,2,2)
stats.probplot(df_feature["osrm_time_sum"], plot= plt, dist="norm")
plt.title('Q-Q plot for aggregation of osrm time (Normality Check)', fontsize=10)
plt.show()
```



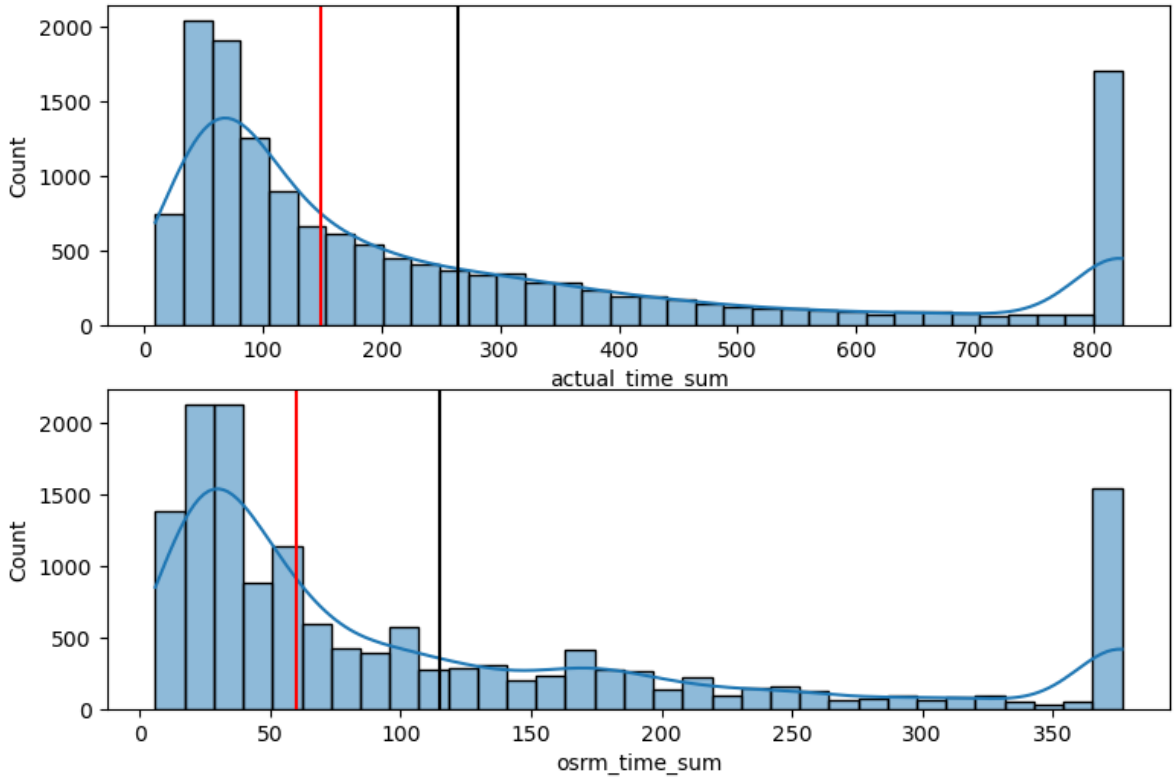
From above tests, it can be concluded that both samples have different variances.
and both samples are not following normal distribution.
Do visual testing to check which distribution these samples are following

visual testing

Checking distribution of samples

In [244…

```
plt.figure(figsize=(9,6))
plt.subplot(2,1,1)
sns.histplot(x=df_feature["actual_time_sum"],kde=True)
plt.axvline(np.mean(df_feature["actual_time_sum"]), c="k");
plt.axvline(np.quantile(df_feature["actual_time_sum"],0.5), c="r")

plt.subplot(2,1,2)
sns.histplot(x=df_feature["osrm_time_sum"],kde=True)
plt.axvline(np.mean(df_feature["osrm_time_sum"]), c="k");
plt.axvline(np.quantile(df_feature["osrm_time_sum"],0.5), c="r")
plt.show()
```



from above plots, it can be inferred that distribution is right tailed and highly skewed towards right.
There is huge outlier at extreme right tail.
verify this mathmematically by using function skew

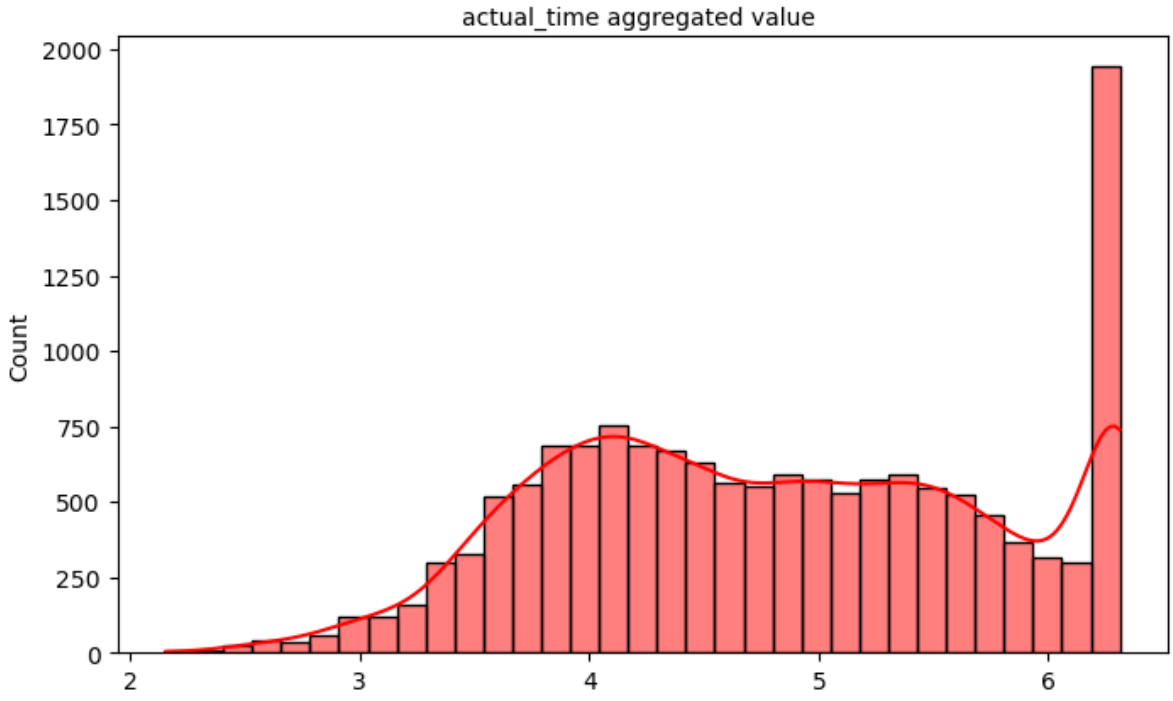
Checking Skewness

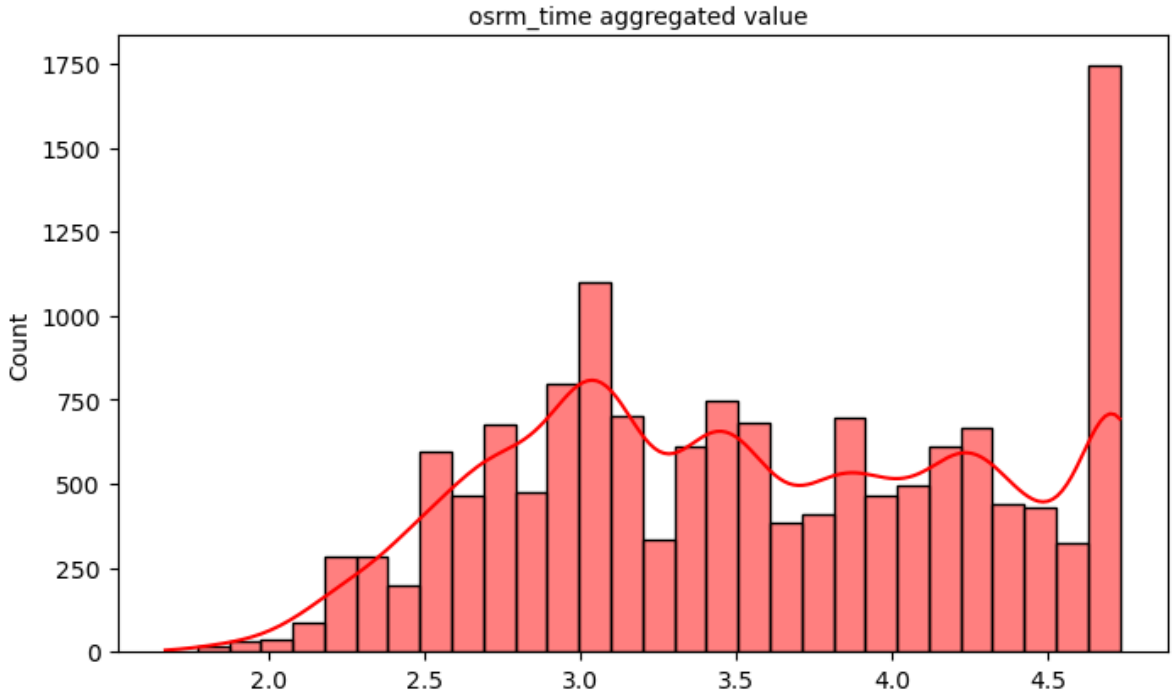
In [244…
```
(skew(df_feature["actual_time_sum"]),skew(df_feature["osrm_time_sum"]))
```

Out[2441]: (1.1734145638697677, 1.237097403518547)

As the value of skew is positive, it can be inferred that the distribution is skewed towards right.
Or it is right tailed.
As observed from graphs, samples does not follow normal distribution,
so taking into consideration, apply BOXCOX transform to get normally distributed samples

In [244…
```
# BOX-COX Transform
actual_agg= stats.boxcox(df_feature["actual_time_sum"])[0]
osrm_agg= stats.boxcox(df_feature["osrm_time_sum"])[0]

fig, axis = plt.subplots(nrows=2, figsize=(8,9))
fig.subplots_adjust(top=1)
sns.histplot(data= actual_agg,kde= True,ax= axis[0], color = "red")
axis[0].set_title("actual_time aggregated value ", pad=5, fontsize=10)
sns.histplot(data= osrm_agg,kde= True,ax= axis[1], color = "red")
axis[1].set_title("osrm_time aggregated value ", pad=5, fontsize=10)
plt.show()
```





In [244…

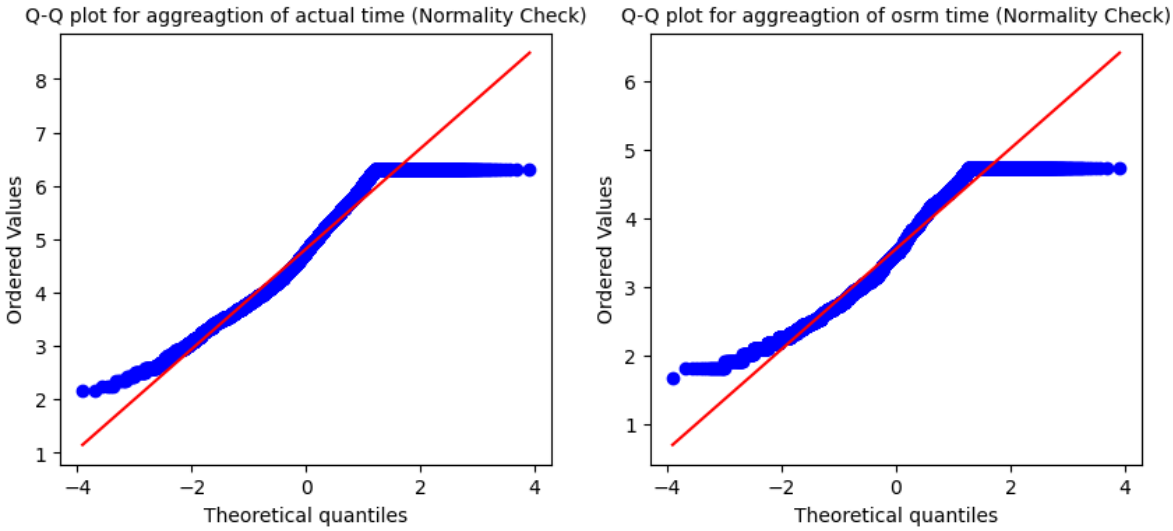
```
# checking distribution of normality using shapiro wilk test
# Null hypothesis(Ho) : Sample is from the normal distributions
# Alternate Hypothesis(Ha): Sample is not from the normal distributions.
shapiro_stat, p_value =stats.shapiro(actual_agg)
print(shapiro_stat, p_value)
result(p_value)
shapiro_stat, p_value =stats.shapiro(osrm_agg)
print(shapiro_stat, p_value)
result(p_value)
```

```
0.9610450863838196 0.0
Reject Null Hypothesis
0.9579086899757385 0.0
Reject Null Hypothesis
```

Even after boxcox transform, shapiro test was rejecting null hypothesis.

It is to noted here that when your sample size is large, even the smallest deviation
from normality will lead to a rejected null.
Therefore, there is need to check normality visually by using QQ plots.

In [244…
```
#Construct QQ plots
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
stats.probplot(actual_agg, plot= plt, dist="norm")
plt.title('Q-Q plot for aggreagtion of actual time (Normality Check)', fontsize=10
plt.subplot(1,2,2)
stats.probplot(osrm_agg, plot= plt, dist="norm")
plt.title('Q-Q plot for aggreagtion of osrm time (Normality Check)', fontsize=10)
plt.show()
```



The significance level of the t-test is somewhat robust against mild to moderate deviations
from normality, but we must take care not to claim too much.

performing t-test on samples

In [244…
```
# 2-sample t_stat test
t_stat, p_value = stats.ttest_ind(actual_agg,osrm_agg)
print("t_stat :",t_stat,"p_value :",p_value)
result(p_value)
```

```
t_stat : 126.63730138397446 p_value : 0.0
Reject Null Hypothesis
```

As p< α we have enough evidence to reject the null hypothesis that is, there is a difference
between the mean for actual time aggregated value & osrm time aggregated value.

Performing Correlation test on samples

In [244…
```
# H0: No relation between time_diff and scan_diff
# Ha: relation between time_diff and scan_diff
pe_stat,p_value = pearsonr(x = actual_agg, y = osrm_agg )
```

```
print(pe_stat,p_value)
result(p_value)
```

```
0.9092262185417299 0.0
Reject Null Hypothesis
```

```
Correlation test tells whether there is relation between those
fields or not. They don't convey
the nature of relation.
There is relation between actual time aggregated value and osrm
time aggregated value
```

4. Do hypothesis testing/ visual analysis between actual_time aggregated value and segment actual time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

APPROACH

Step-1 : Hypothesis formulation

1. Null Hypothesis (H0) - mean of "actual_time aggregated value" and "segment actual time aggregated value" are same.
2. Alternate Hypothesis (HA) - mean of "actual_time aggregated value" and "segment actual time aggregated value" are not same.

Step-2 : Checking for basic assumpitons for the hypothesis

1. Distribution check using QQ Plot
2. Homogeneity of Variances using Lavene's test
3. Check normal distributions using shapiro wilk test

Step-3 : Define Test statistics; Distribution of T under H0.

1. Observe test statistic while performing a T-Test follows Tdistribution.

Step-4: Decide the kind of test.

1. Perform Two tailed t-test

Step-5: Compute the p-value and fix value of alpha.

1. compute t-test value using the ttest function using scipy.stats. set *alpha to be 0.05*

Step-6: Compare p-value and alpha.

1. Based on p-value, accept or reject H0.
   A. p-val > alpha : Accept H0
   B. p-val < alpha : Reject H0

Homogeneity of Variances using Lavene's test

In [244…

```
# Checking Homogeneity of Variances using Lavene's test
# Null hypothesis(Ho) : Samples have similar variances
```

```
# Alternate Hypothesis(Ha) : samples have different variances
levene_stat, p_value =stats.levene(df_feature["actual_time_sum"] ,df_feature["segm
print(levene_stat, p_value)
result(p_value)
```

```
0.33125497574035107 0.5649254161381606
Failed to Reject Null Hypothesis
```

Check normal distributions using shapiro wilk test

In [244…

```
# checking distribution of normality using shapiro wilk test
# Null hypothesis(Ho) : Sample is from the normal distributions
# Alternate Hypothesis(Ha): Sample is not from the normal distributions.
shapiro_stat, p_value =stats.shapiro(df_feature["actual_time_sum"])
print(shapiro_stat, p_value)
result(p_value)
shapiro_stat, p_value =stats.shapiro(df_feature["segment_actual_time_sum_sum"])
print(shapiro_stat, p_value)
result(p_value)
```

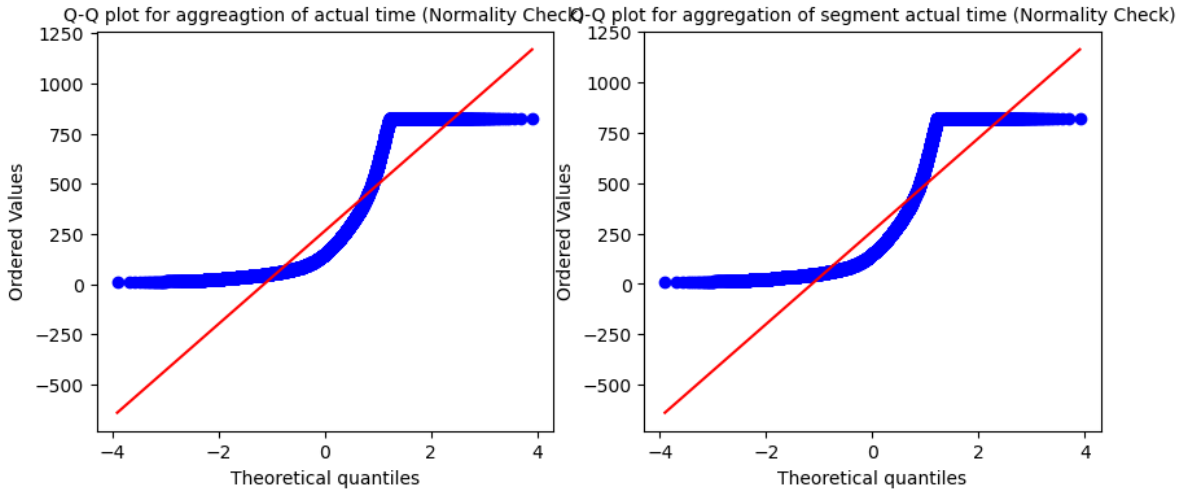
```
0.7887564897537231 0.0
Reject Null Hypothesis
0.7882676720619202 0.0
Reject Null Hypothesis
```

From levene's test, it has been concluded that samples have similar variances.
However, from shapiro wilk test, null hypothesis has been rejected, this means that
data samples does not follow normal distribution
Now, verify the above result using Q-Q plots.

Distribution check using QQ plot

In [244…

```
#Construct QQ plots
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
stats.probplot(df_feature["actual_time_sum"], plot= plt, dist="norm")
plt.title('Q-Q plot for aggreagtion of actual time (Normality Check)', fontsize=10
plt.subplot(1,2,2)
stats.probplot(df_feature["segment_actual_time_sum_sum"], plot= plt, dist="norm")
plt.title('Q-Q plot for aggregation of segment actual time (Normality Check)', fon
plt.show()
```

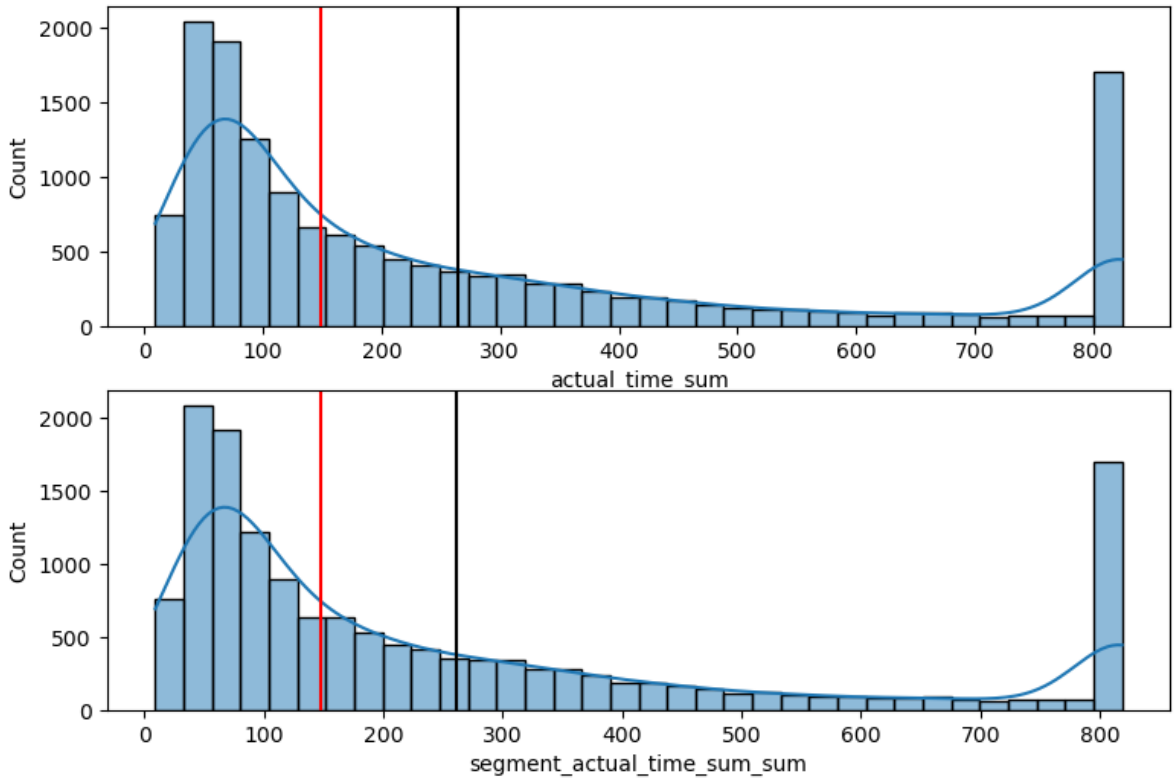


The samples are highly deviated from normal distribution. We can apply boxcox transform

visual testing

Checking distribution of samples

In [245…
```
plt.figure(figsize=(9,6))
plt.subplot(2,1,1)
sns.histplot(x=df_feature["actual_time_sum"],kde=True)
plt.axvline(np.mean(df_feature["actual_time_sum"]), c="k");
plt.axvline(np.quantile(df_feature["actual_time_sum"],0.5), c="r")

plt.subplot(2,1,2)
sns.histplot(x=df_feature["segment_actual_time_sum_sum"],kde=True)
plt.axvline(np.mean(df_feature["segment_actual_time_sum_sum"]), c="k");
plt.axvline(np.quantile(df_feature["segment_actual_time_sum_sum"],0.5), c="r")
plt.show()
```



from above plots, it can be inferred that distribution is right tailed and highly skewed towards right.
There is huge outlier at extreme right tail.
verify this mathmematically by using function skew

Checking Skewness

In [245…
```
(skew(df_feature["actual_time_sum"]),skew(df_feature["segment_actual_time_sum_sum"
```

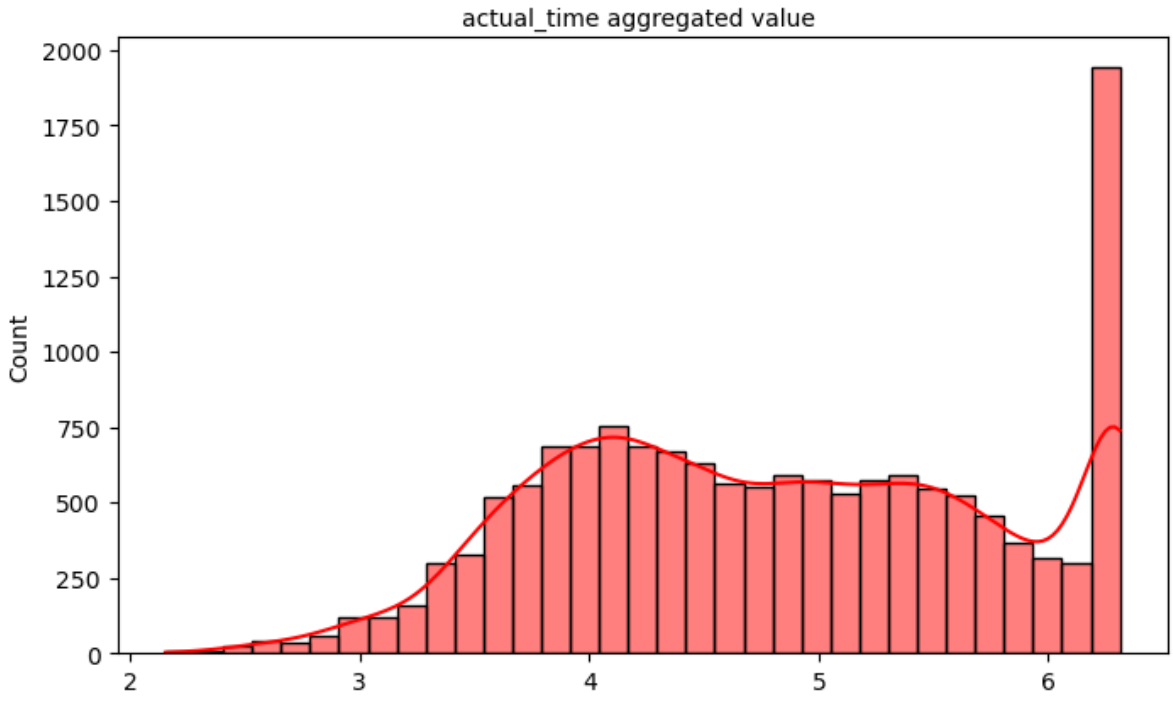
Out[2451]: (1.1734145638697677, 1.174845609188422)

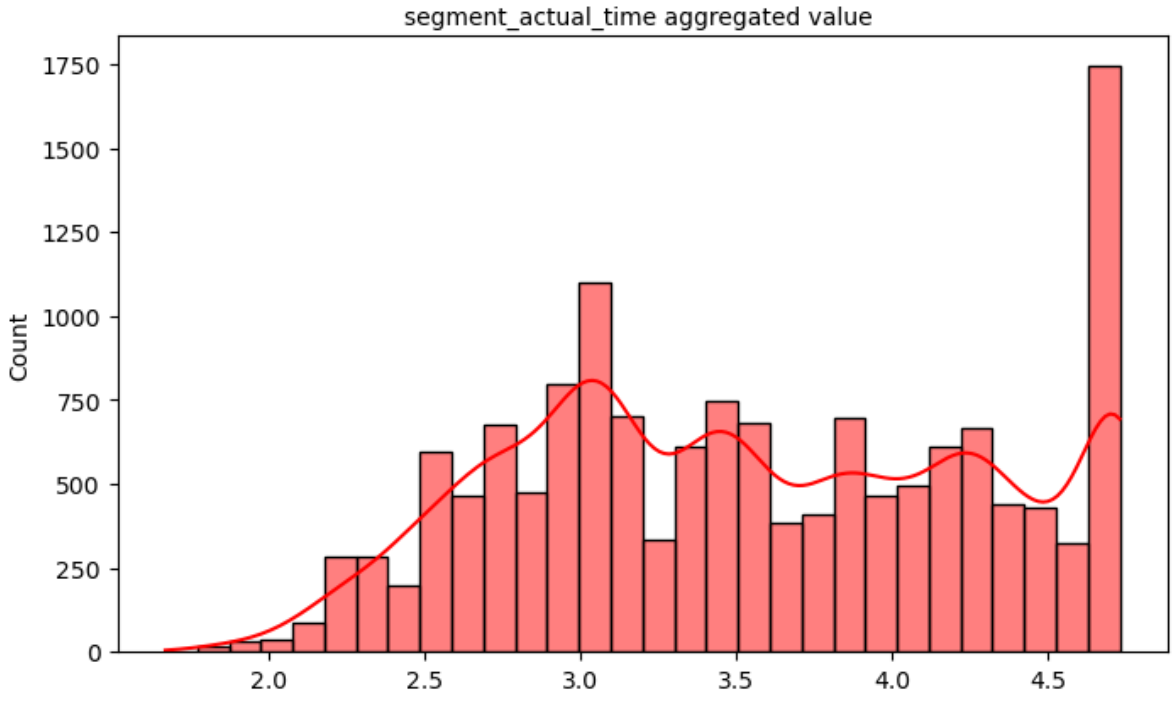
As the value of skew is positive, it can be inferred that the distribution is skewed towards right.

Or it is right tailed.
As observed from graphs, samples does not follow normal distribution,
so taking into consideration, apply BOXCOX transform to get normally distributed samples

In [245…

```
# BOX-COX Transform
actual_agg= stats.boxcox(df_feature["actual_time_sum"])[0]
seg_actual_agg= stats.boxcox(df_feature["osrm_time_sum"])[0]

fig, axis = plt.subplots(nrows=2, figsize=(8,9))
fig.subplots_adjust(top=1)
sns.histplot(data= actual_agg,kde= True,ax= axis[0], color = "red")
axis[0].set_title("actual_time aggregated value ", pad=5, fontsize=10)
sns.histplot(data= seg_actual_agg,kde= True,ax= axis[1], color = "red")
axis[1].set_title("segment_actual_time aggregated value ", pad=5, fontsize=10)
plt.show()
```





In [245…

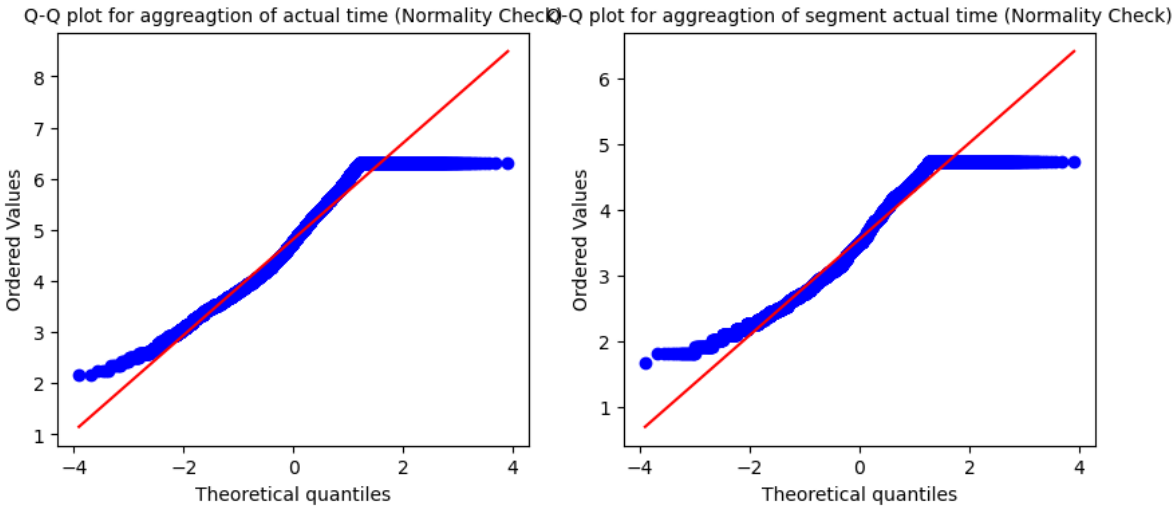
```
# checking distribution of normality using shapiro wilk test
# Null hypothesis(Ho) : Sample is from the normal distributions
# Alternate Hypothesis(Ha): Sample is not from the normal distributions.
shapiro_stat, p_value =stats.shapiro(actual_agg)
print(shapiro_stat, p_value)
result(p_value)
shapiro_stat, p_value =stats.shapiro(seg_actual_agg)
print(shapiro_stat, p_value)
result(p_value)
```

```
0.9610450863838196 0.0
Reject Null Hypothesis
0.9579086899757385 0.0
Reject Null Hypothesis
```

Even after boxcox transform, shapiro test was rejecting null hypothesis.
It is to noted here that when your sample size is large, even the smallest deviation
from normality will lead to a rejected null.
Therefore, there is need to check normality visually by using QQ plots.

In [245…

```
#Construct QQ plots
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
stats.probplot(actual_agg, plot= plt, dist="norm")
plt.title('Q-Q plot for aggreagtion of actual time (Normality Check)', fontsize=10
plt.subplot(1,2,2)
stats.probplot(seg_actual_agg, plot= plt, dist="norm")
plt.title('Q-Q plot for aggreagtion of segment actual time (Normality Check)', fon
plt.show()
```



The significance level of the t-test is somewhat robust against mild to moderate deviations
from normality, but we must take care not to claim too much.

performing t-test on samples

In [245…

```
# 2-sample t_stat test
t_stat, p_value = stats.ttest_ind(actual_agg,seg_actual_agg)
```

```
print("t_stat :",t_stat,"p_value :",p_value)
result(p_value)
```

```
t_stat : 126.63730138397446 p_value : 0.0
Reject Null Hypothesis
```

As p< α we have enough evidence to reject the null hypothesis that is, there is a difference
between the mean for actual time aggregated value & osrm time aggregated value.

Performing Correlation test on samples

In [245…

```
pe_stat,p_value = pearsonr(x = df_feature["actual_time_sum"], y = df_feature["segm
print(pe_stat,p_value)
result(p_value)
```

```
0.9999863044565497 0.0
Reject Null Hypothesis
```

Correlation test tells whether there is relation between those fields or not. They don't convey
the nature of relation.
There is relation between actual time aggregated value and segment actual time aggregated value

5. Do hypothesis testing/ visual analysis between osrm distance aggregated value and segment osrm distance aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

APPROACH

Step-1 : Hypothesis formulation

1. Null Hypothesis (H0) - mean of " osrm distance aggregated value" and " osrm distance aggregated value" are same.
2. Alternate Hypothesis (HA) - mean of " osrm distance aggregated value" and " osrm distance aggregated value" are not same.

Step-2 : Checking for basic assumpitons for the hypothesis

1. Distribution check using QQ Plot
2. Homogeneity of Variances using Lavene's test
3. Check normal distributions using shapiro wilk test

Step-3 : Define Test statistics; Distribution of T under H0.

1. Observe test statistic while performing a T-Test follows Tdistribution.

Step-4: Decide the kind of test.

1. Perform Two tailed t-test

Step-5: Compute the p-value and fix value of alpha.

1. compute t-test value using the ttest function using scipy.stats. set *alpha to be 0.05*

Step-6: Compare p-value and alpha.

1. Based on p-value, accept or reject H0.
   A. p-val > alpha : Accept H0
   B. p-val < alpha : Reject H0

Homogeneity of Variances using Lavene's test

In [245...
```
# Checking Homogeneity of Variances using Lavene's test
# Null hypothesis(Ho) : Samples have similar variances
# Alternate Hypothesis(Ha) : samples have different variances
levene_stat, p_value =stats.levene(df_feature["osrm_distance_sum"] ,df_feature["seg
print(levene_stat, p_value)
result(p_value)
```

```
15.608115475223101 7.809983647427559e-05
Reject Null Hypothesis
```

Check normal distributions using shapiro wilk test

In [245...
```
# checking distribution of normality using shapiro wilk test
# Null hypothesis(Ho) : Sample is from the normal distributions
# Alternate Hypothesis(Ha): Sample is not from the normal distributions.
shapiro_stat, p_value =stats.shapiro(df_feature["osrm_distance_sum"])
print(shapiro_stat, p_value)
result(p_value)
shapiro_stat, p_value =stats.shapiro(df_feature["segment_osrm_distance_sum_sum"])
print(shapiro_stat, p_value)
result(p_value)
```

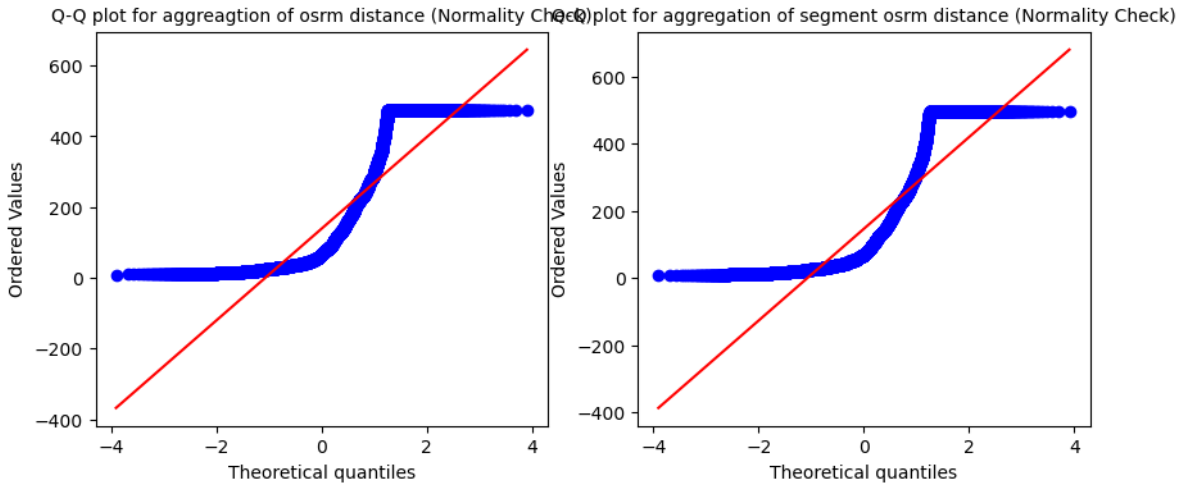
```
0.7694376111030579 0.0
Reject Null Hypothesis
0.774215579032898 0.0
Reject Null Hypothesis
```

```
From levene's test, it has been concluded that samples do not have
similar variances.
From shapiro wilk test, null hypothesis has been rejected, this
means that
data samples does not follow normal distribution
Now, verify the above result using Q-Q plots.
```

Distribution check using QQ plot

In [245...
```
#Construct QQ plots
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
stats.probplot(df_feature["osrm_distance_sum"], plot= plt, dist="norm")
plt.title('Q-Q plot for aggreagtion of osrm distance (Normality Check)', fontsize=
plt.subplot(1,2,2)
stats.probplot(df_feature["segment_osrm_distance_sum_sum"], plot= plt, dist="norm"
```

```
plt.title('Q-Q plot for aggregation of segment osrm distance (Normality Check)', f
plt.show()
```



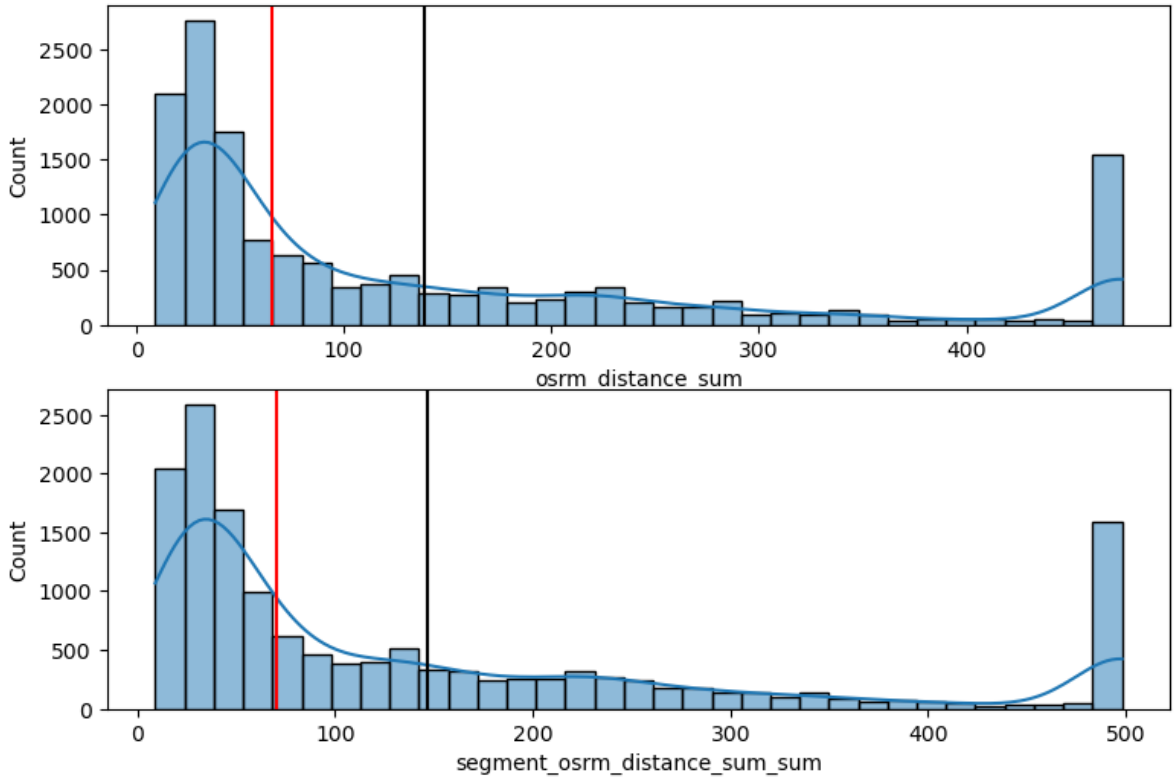
The samples are highly deviated from normal distribution. We can apply boxcox transform

visual testing

Checking distribution of samples

In [246...

```
plt.figure(figsize=(9,6))
plt.subplot(2,1,1)
sns.histplot(x=df_feature["osrm_distance_sum"],kde=True)
plt.axvline(np.mean(df_feature["osrm_distance_sum"]), c="k");
plt.axvline(np.quantile(df_feature["osrm_distance_sum"],0.5), c="r")

plt.subplot(2,1,2)
sns.histplot(x=df_feature["segment_osrm_distance_sum_sum"],kde=True)
plt.axvline(np.mean(df_feature["segment_osrm_distance_sum_sum"]), c="k");
plt.axvline(np.quantile(df_feature["segment_osrm_distance_sum_sum"],0.5), c="r")
plt.show()
```



from above plots, it can be inferred that distribution is right tailed and highly skewed towards right.
There is huge outlier at extreme right tail.
verify this mathmematically by using function skew

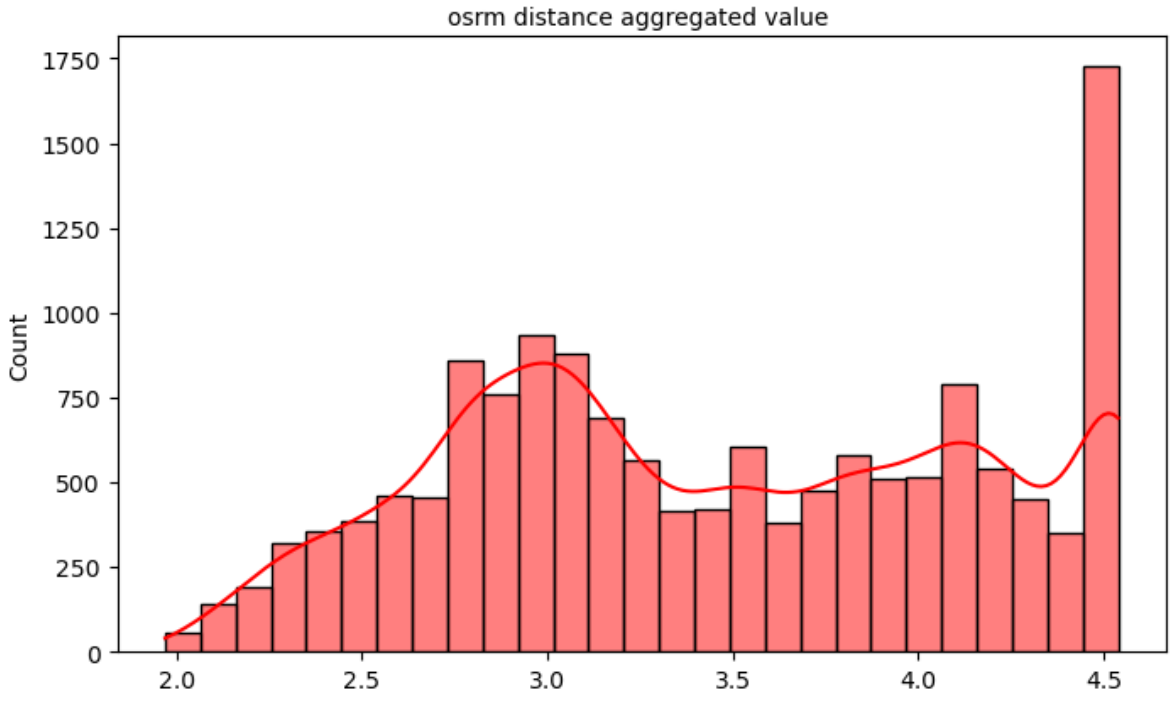
Checking Skewness

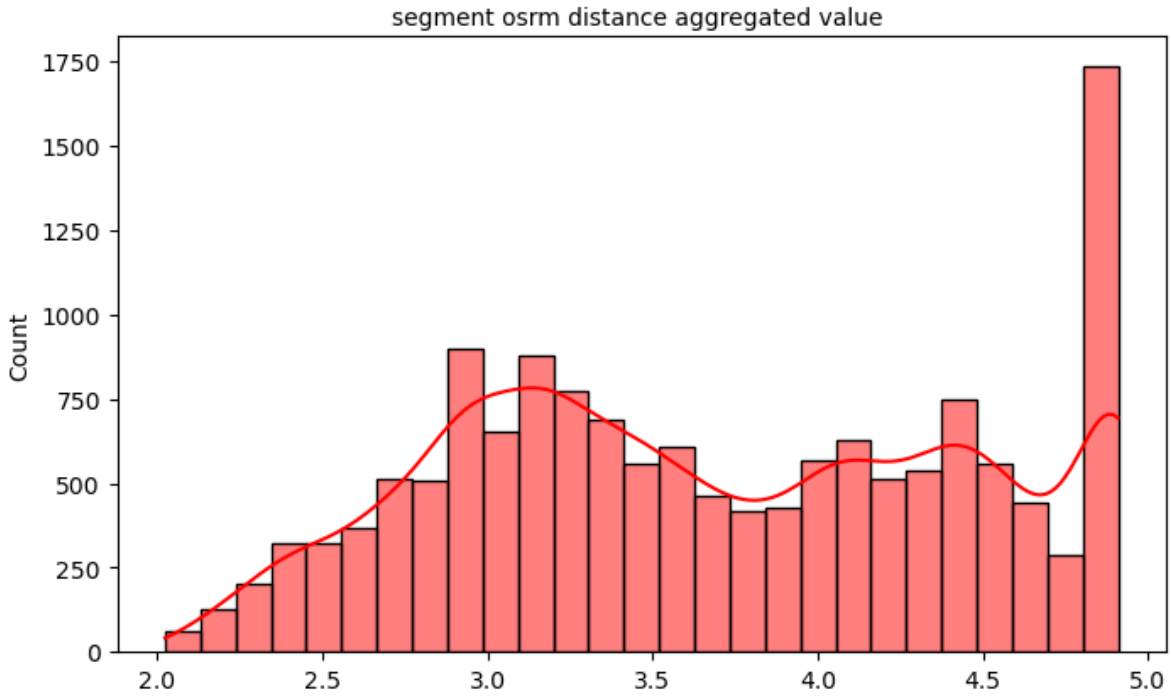
In [246…
```
(skew(df_feature["osrm_distance_sum"]),skew(df_feature["segment_osrm_distance_sum_s
```

Out[2461]: (1.2776592590562519, 1.2634575002820783)

As the value of skew is positive, it can be inferred that the distribution is skewed towards right.
Or it is right tailed.
As observed from graphs, samples does not follow normal distribution,
so taking into consideration, apply BOXCOX transform to get normally distributed samples

In [246…
```
# BOX-COX Transform
dis_agg= stats.boxcox(df_feature["osrm_distance_sum"])[0]
seg_dis_agg= stats.boxcox(df_feature["segment_osrm_distance_sum_sum"])[0]

fig, axis = plt.subplots(nrows=2, figsize=(8,9))
fig.subplots_adjust(top=1)
sns.histplot(data= dis_agg,kde= True,ax= axis[0], color = "red")
axis[0].set_title("osrm distance aggregated value ", pad=5, fontsize=10)
sns.histplot(data= seg_dis_agg,kde= True,ax= axis[1], color = "red")
axis[1].set_title("segment osrm distance aggregated value ", pad=5, fontsize=10)
plt.show()
```





In [246…

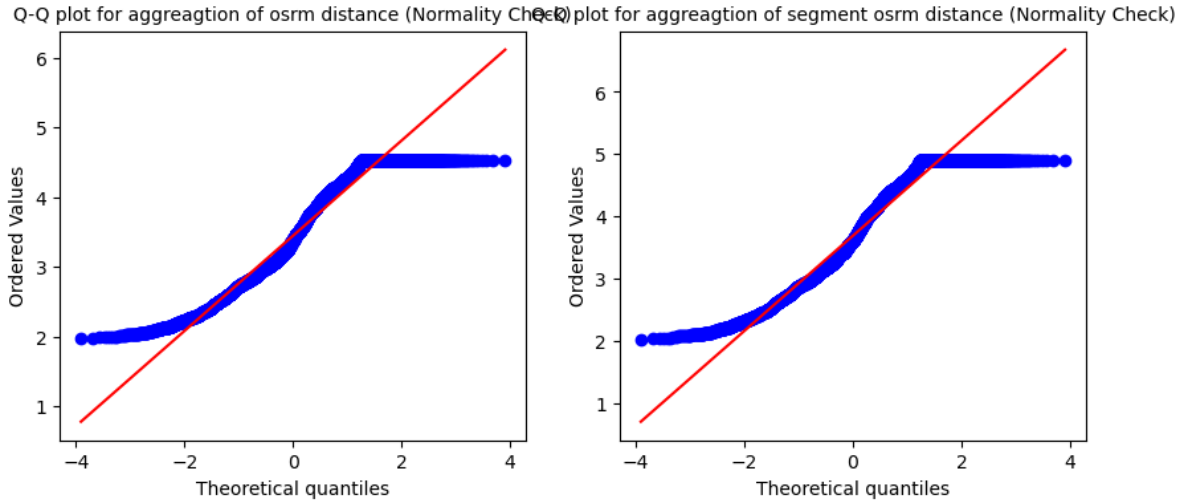
```
# checking distribution of normality using shapiro wilk test
# Null hypothesis(Ho) : Sample is from the normal distributions
# Alternate Hypothesis(Ha): Sample is not from the normal distributions.
shapiro_stat, p_value =stats.shapiro(dis_agg)
print(shapiro_stat, p_value)
result(p_value)
shapiro_stat, p_value =stats.shapiro(seg_dis_agg)
print(shapiro_stat, p_value)
result(p_value)
```

```
0.9513328671455383 0.0
Reject Null Hypothesis
0.9534818530082703 0.0
Reject Null Hypothesis
```

Even after boxcox transform, shapiro test was rejecting null hypothesis.

It is to noted here that when your sample size is large, even the smallest deviation
from normality will lead to a rejected null.
Therefore, there is need to check normality visually by using QQ plots.

In [246...
```
#Construct QQ plots
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
stats.probplot(dis_agg, plot= plt, dist="norm")
plt.title('Q-Q plot for aggreagtion of osrm distance (Normality Check)', fontsize=
plt.subplot(1,2,2)
stats.probplot(seg_dis_agg, plot= plt, dist="norm")
plt.title('Q-Q plot for aggreagtion of segment osrm distance (Normality Check)', f
plt.show()
```



The significance level of the t-test is somewhat robust against mild to moderate deviations
from normality, but we must take care not to claim too much.

performing t-test on samples

In [246...
```
# 2-sample t_stat test
t_stat, p_value = stats.ttest_ind(dis_agg,seg_dis_agg)
print("t_stat :",t_stat,"p_value :",p_value)
result(p_value)
```

```
t_stat : -27.766072502702688 p_value : 1.5613404576195671e-167
Reject Null Hypothesis
```

As p< α we have enough evidence to reject the null hypothesis that is, there is a difference
between the mean for actual time aggregated value & osrm time aggregated value.

Performing Correlation test on samples

In [246...
```
pe_stat,p_value = pearsonr(x = df_feature["osrm_distance_sum"], y = df_feature["se
print(pe_stat,p_value)
result(p_value)
```

```
0.992302952984983 0.0
Reject Null Hypothesis
```

```
Correlation test tells whether there is relation between those
fields or not. They don't convey
the nature of relation.
There is relation between actual time aggregated value and segment
actual time aggregated value
```

6. Do hypothesis testing/ visual analysis between osrm time aggregated value and segment osrm time aggregated value (aggregated values are the values you'll get after merging the rows on the basis of trip_uuid)

APPROACH

Step-1 : Hypothesis formulation

1. Null Hypothesis (H0) - mean of "osrm_time_sum" and "segment_osrm_time_sum_sum" are same.
2. Alternate Hypothesis (HA) - mean of "osrm_time_sum" and "segment_osrm_time_sum_sum" are not same.

Step-2 : Checking for basic assumpitons for the hypothesis

1. Distribution check using QQ Plot
2. Homogeneity of Variances using Lavene's test
3. Check normal distributions using shapiro wilk test

Step-3 : Define Test statistics; Distribution of T under H0.

1. Observe test statistic while performing a T-Test follows Tdistribution.

Step-4: Decide the kind of test.

1. Perform Two tailed t-test

Step-5: Compute the p-value and fix value of alpha.

1. compute t-test value using the ttest function using scipy.stats. set *alpha to be 0.05*

Step-6: Compare p-value and alpha.

1. Based on p-value, accept or reject H0.
    A. p-val > alpha : Accept H0
    B. p-val < alpha : Reject H0

Homogeneity of Variances using Lavene's test

In [246…

```
# Checking Homogeneity of Variances using Lavene's test
# Null hypothesis(Ho) : Samples have similar variances
```

```
# Alternate Hypothesis(Ha) : samples have different variances
levene_stat, p_value =stats.levene(df_feature["osrm_time_sum"] ,df_feature["segmen
print(levene_stat, p_value)
result(p_value)
```

```
59.07047822994247 1.568182390292768e-14
Reject Null Hypothesis
```

Check normal distributions using shapiro wilk test

In [246...

```
# checking distribution of normality using shapiro wilk test
# Null hypothesis(Ho) : Sample is from the normal distributions
# Alternate Hypothesis(Ha): Sample is not from the normal distributions.
shapiro_stat, p_value =stats.shapiro(df_feature["osrm_time_sum"])
print(shapiro_stat, p_value)
result(p_value)
shapiro_stat, p_value =stats.shapiro(df_feature["segment_osrm_time_sum_sum"])
print(shapiro_stat, p_value)
result(p_value)
```

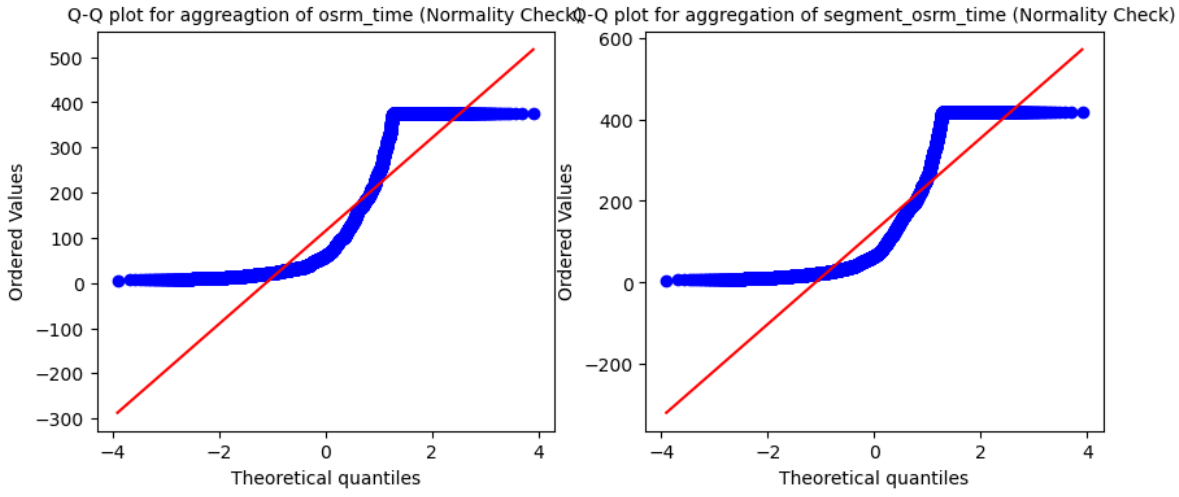
```
0.7823594808578491 0.0
Reject Null Hypothesis
0.7852588295936584 0.0
Reject Null Hypothesis
```

From levene's test, it has been concluded that samples do not have similar variances.
However, from shapiro wilk test, null hypothesis has been rejected, this means that
data samples does not follow normal distribution
Now, verify the above result using Q-Q plots.

Distribution check using QQ plot

In [246...

```
#Construct QQ plots
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
stats.probplot(df_feature["osrm_time_sum"], plot= plt, dist="norm")
plt.title('Q-Q plot for aggreagtion of osrm_time (Normality Check)', fontsize=10)
plt.subplot(1,2,2)
stats.probplot(df_feature["segment_osrm_time_sum_sum"], plot= plt, dist="norm")
plt.title('Q-Q plot for aggregation of segment_osrm_time (Normality Check)', fonts
plt.show()
```

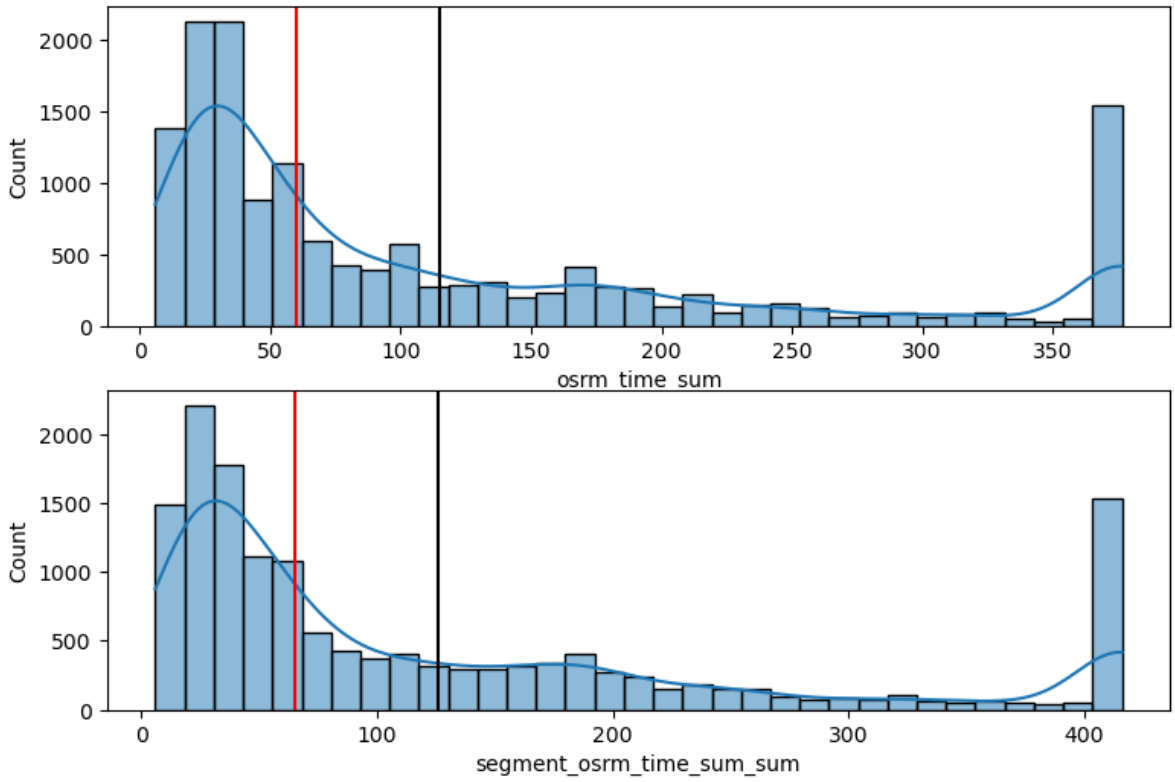


The samples are highly deviated from normal distribution. We can apply boxcox transform

visual testing

Checking distribution of samples

In [247…
```
plt.figure(figsize=(9,6))
plt.subplot(2,1,1)
sns.histplot(x=df_feature["osrm_time_sum"],kde=True)
plt.axvline(np.mean(df_feature["osrm_time_sum"]), c="k");
plt.axvline(np.quantile(df_feature["osrm_time_sum"],0.5), c="r")

plt.subplot(2,1,2)
sns.histplot(x=df_feature["segment_osrm_time_sum_sum"],kde=True)
plt.axvline(np.mean(df_feature["segment_osrm_time_sum_sum"]), c="k");
plt.axvline(np.quantile(df_feature["segment_osrm_time_sum_sum"],0.5), c="r")
plt.show()
```



from above plots, it can be inferred that distribution is right tailed and highly skewed towards right.
There is huge outlier at extreme right tail.
verify this mathmematically by using function skew

Checking Skewness

In [247…
```
(skew(df_feature["osrm_time_sum"]),skew(df_feature["segment_osrm_time_sum_sum"]))
```

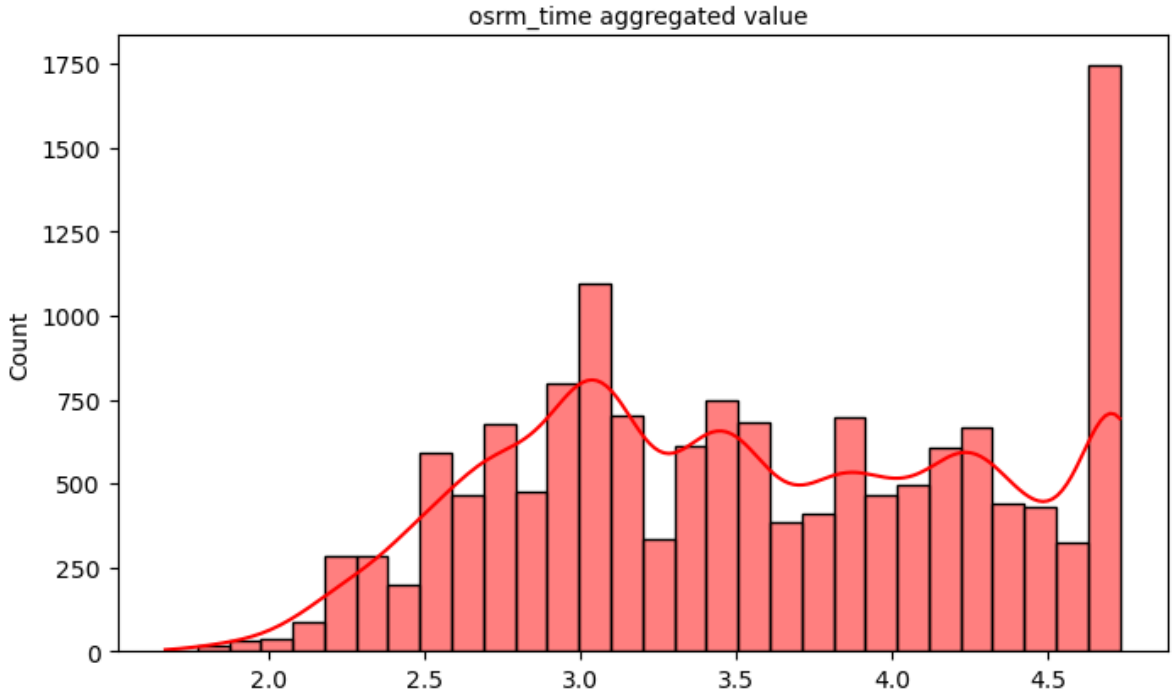
Out[2471]: (1.237097403518547, 1.237792741535024)

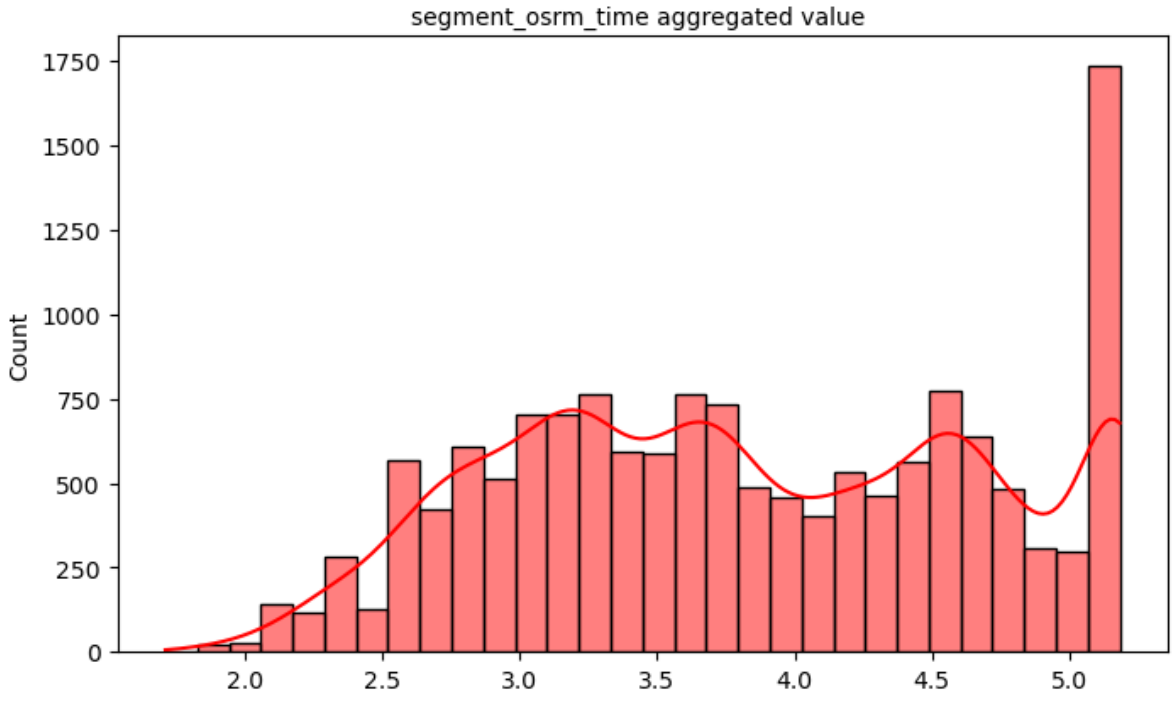
As the value of skew is positive, it can be inferred that the distribution is skewed towards right.

Or it is right tailed.
As observed from graphs, samples does not follow normal distribution,
so taking into consideration, apply BOXCOX transform to get normally distributed samples

In [247…

```
# BOX-COX Transform
actual_agg= stats.boxcox(df_feature["osrm_time_sum"])[0]
seg_actual_agg= stats.boxcox(df_feature["segment_osrm_time_sum_sum"])[0]

fig, axis = plt.subplots(nrows=2, figsize=(8,9))
fig.subplots_adjust(top=1)
sns.histplot(data= actual_agg,kde= True,ax= axis[0], color = "red")
axis[0].set_title("osrm_time aggregated value ", pad=5, fontsize=10)
sns.histplot(data= seg_actual_agg,kde= True,ax= axis[1], color = "red")
axis[1].set_title("segment_osrm_time aggregated value ", pad=5, fontsize=10)
plt.show()
```





In [247…

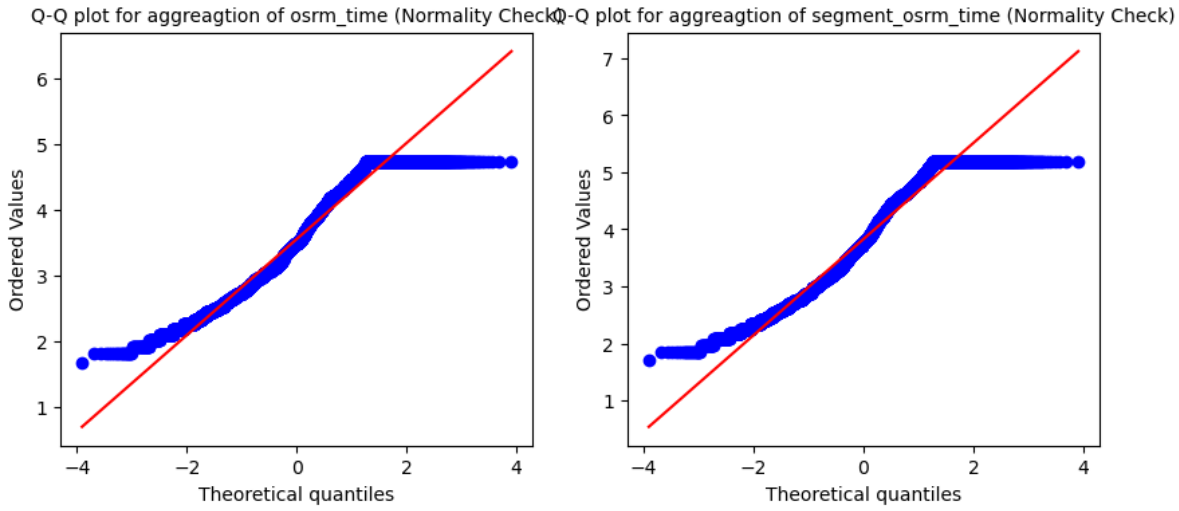
```
# checking distribution of normality using shapiro wilk test
# Null hypothesis(Ho) : Sample is from the normal distributions
# Alternate Hypothesis(Ha): Sample is not from the normal distributions.
shapiro_stat, p_value =stats.shapiro(actual_agg)
print(shapiro_stat, p_value)
result(p_value)
shapiro_stat, p_value =stats.shapiro(seg_actual_agg)
print(shapiro_stat, p_value)
result(p_value)
```

```
0.9579086899757385 0.0
Reject Null Hypothesis
0.9584182500839233 0.0
Reject Null Hypothesis
```

Even after boxcox transform, shapiro test was rejecting null hypothesis.
It is to noted here that when your sample size is large, even the smallest deviation
from normality will lead to a rejected null.
Therefore, there is need to check normality visually by using QQ plots.

In [247…

```
#Construct QQ plots
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
stats.probplot(actual_agg, plot= plt, dist="norm")
plt.title('Q-Q plot for aggreagtion of osrm_time (Normality Check)', fontsize=10)
plt.subplot(1,2,2)
stats.probplot(seg_actual_agg, plot= plt, dist="norm")
plt.title('Q-Q plot for aggreagtion of segment_osrm_time (Normality Check)', fonts
plt.show()
```



The significance level of the t-test is somewhat robust against mild to moderate deviations
from normality, but we must take care not to claim too much.

performing t-test on samples

In [247…

```
# 2-sample t_stat test
t_stat, p_value = stats.ttest_ind(actual_agg,seg_actual_agg)
```

```
print("t_stat :",t_stat,"p_value :",p_value)
result(p_value)
```

```
t_stat : -29.296089122725352 p_value : 5.2460723299703675e-186
Reject Null Hypothesis
```

As p< α we have enough evidence to reject the null hypothesis that is, there is a difference
between the mean for actual time aggregated value & osrm time aggregated value.

Performing Correlation test on samples

In [247…

```
pe_stat,p_value = pearsonr(x = df_feature["osrm_time_sum"], y = df_feature["segmen
print(pe_stat,p_value)
result(p_value)
```

```
0.989576366295789 0.0
Reject Null Hypothesis
```

Correlation test tells whether there is relation between those fields or not. They don't convey
the nature of relation.
There is relation between actual time aggregated value and segment actual time aggregated value

Encoding data for imputation

Do one-hot encoding of categorical variables (like route_type)?

In [247…

```
one_hot_encoded_data = pd.get_dummies(df_feature, columns = ['data','route_type'])
display(one_hot_encoded_data.iloc[:,25:])
```

	trip_creation_year	trip_creation_month	trip_creation_day	data_test	data_training	route_ty
0	2018	9	12	0	1	
1	2018	9	12	0	1	
2	2018	9	12	0	1	
3	2018	9	12	0	1	
4	2018	9	12	0	1	
...	...	...	...	...	...	
14812	2018	10	3	1	0	
14813	2018	10	3	1	0	
14814	2018	10	3	1	0	
14815	2018	10	3	1	0	
14816	2018	10	3	1	0	

14817 rows × 7 columns

Column Normalization /Column Standardization

In [247…

```
df_feature.iloc[:,20:].head(2)
```

Out[2478]:

	actual_distance_to_destination_sum	actual_time_sum	osrm_time_sum	osrm_distance_sum	segn
0	377.202161	824.5	376.5	474.9587	
1	73.186911	143.0	68.0	85.1110	

In [247…

```
#import associated libraries
from sklearn.preprocessing import StandardScaler, MinMaxScaler

df_feature_num = df_feature.loc[:,['start_scan_to_end_scan_sum','actual_distance_t
                                   'actual_time_sum','osrm_time_sum','osrm_distance_
                                   'segment_actual_time_sum_sum','segment_osrm_dist
                                   'segment_osrm_time_sum_sum']]

scaler = StandardScaler()
data_norm = scaler.fit_transform(df_feature_num)
data_norm = pd.DataFrame(df_feature_num, columns=df_feature_num.columns)
data_norm.head()
```

Out[2479]:

	start_scan_to_end_scan_sum	actual_distance_to_destination_sum	actual_time_sum	osrm_time_su
0	1369.0	377.202161	824.5	376
1	180.0	73.186911	143.0	68
2	1369.0	377.202161	824.5	376
3	100.0	17.175274	59.0	15
4	717.0	127.448500	341.0	117

Insights

1. Shape of the data is (144867, 24).
2. Data fetch if from sepetember 2018 to October 2018.
3. The entire data is heavily right skewed.
4. 60% data is from Carting route_type & the remaining 40% is from FTL route type.
5. 88% of the trips are from October Month & remaining from November.
6. Order were made less in starting & ending dates of the months compared to mid of months. Though the difference is not huge.
7. Maximum orders are coming from Maharashtra state followed by Karnataka state.
8. Maximum order are coming from Bangalore city.
9. Least orders are coming from North-East states.

Results of Testing

1. There is no difference in the means of time taken between od_start_time and od_end_time and start_scan_to_end_scan. However, the variances of both samples are similar.
2. There is a difference between the means of aggregated actual time & aggregated osrm time. Mean of aggreagted actual_time is higher than that of aggregated osrm time
3. There is a difference between the means of aggregated actual time & aaggregated segment actual time. Mean of aggregated actual time is higher than aggregated segment actual time
4. There is a difference between the means of aggregated osrm distance & aggregated segment osrm distance.
5. There is a difference between the means of aggregated osrm time & aggregated segment osrm time.

Recommendation

1. Carting taking much time to deliver. The speed of Carting needs to be improved.

2. As highest number of orders are coming from Maharashtra state, there is need to make sure that corriders must not take not much and fastest routes must be available.
3. The number of deliveries are much higher within the city of Banglore. Hence, delivery centres need to be connected with fastest route possible and simulataneously needs more manpower in this city compared to other cities.
4. Highest delivery happened on Wednesdays. Therefore, delivery trips can be assigned in night or early morning to avoid day traffic.
5. Very less orders are coming from North-East states. So, marketing team needs to analyze deeply into this matter.
6. FTL route consists of 40% of the total orders. There is need to have more ways to promote. FTL route handling system can be implemented to increase this percentage
7. OSRM data are very much deviated from actual data. Hence, the already applied sytem is not reliable. There is need to update Recommender system.
8. Storage, warehousing and materials handling need to be best in states like Maharashtra, Karnataka, Haryana.